UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Trček

Računalniški vid za pametno knjižnico

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Trček

Računalniški vid za pametno knjižnico

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Danijel Skočaj

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Sodobne tehnologije omogočajo nove načine opravljanja ustaljenih postopkov, kot je npr. izposoja knjig v knjižnici. Z uporabo barvno-globinske kamere Kinect izdelajte brezdotični uporabniški vmesnik, ki bo omogočal samodejno razpoznavanje knjig. Uporabnik naj sistemu na naraven način pokaže knjigo; sistem naj nato z uporabo slikovne in 3D informacije zazna in segmentira platnico knjige ter jo razpozna. Implementirajte in ovrednotite različne klasifikatorje ter najuspešnejšega integrirajte v brezdotični uporabniški vmesnik.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Matej Trček sem avtor diplomskega dela z naslovom:

Računalniški vid za pametno knjižnico

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Danijela Skočaja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. marca 2016 Podpis avtorja:

Družini.

Kazalo

Seznam uporabljenih kratic

kratica	Angleško	Slovensko
RANSAC	random sample consensus	soglasje naključnih vzorcev
PCA	principle component analysis	analiza glavnih komponent
SVM	support vector machine	metoda podpornih vektorjev
k-NN	k nearest neighbours	k najbližjih sosedov
SIFT	scale invariant feature transform	transformacija značilk z invarianco glede na merilo
RGB	red, green and blue	rdeča, zelena in modra
RGBD	red, green, blue and depth	rdeča, zelena, modra in globina
BGR	blue, green and red	modra, zelena in rdeča

Povzetek

S prostoročnimi vmesniki lahko tehnologijo na udoben način vključimo v vsakdanje dejavnosti. Diplomsko delo opisuje izdelavo aplikacije za prepoznavanje knjižnih platnic z barvno-globinsko kamero za rabo v pametni knjižnici. Aplikacija v globinski sliki zazna ravnino, v njej najde štirikotnik in ga naravna v ravnino kamere. V naslednjem koraku s tehnikami računalniškega vida primerja najdeno sliko s pripravljeno podatkovno bazo posnetkov knjižnih platnic in najde najboljše ujemanje. Globinska slika je zajeta s kamero Microsoft Kinect, aplikacija pa je sprogramirana v jeziku C++ z uporabo knjižnice OpenCV. Za iskanje ravnine se uporablja algoritem RANSAC, za iskanje oglišč pa je bil razvit algoritem, ki se opira na iskanje najbolj oddaljenih točk in preverjanje kotov med njimi. Za opis vzorcev se uporabljata algoritem PCA in histogram pojavitev značilk SIFT, za klasifikacijo pa algoritma SVM in *k*-NN ter evklidska razdalja do najboljšega primera iz vsakega razreda.

Ključne besede: računalniški vid, prepoznavanje slik, globinske slike

Abstract

Hands-free interfaces allow us to comfortably integrate technology into mundane tasks. This thesis describes the development of an application for book cover recognition using an RGBD camera for use in a smart library. The application detects a plane within the depth image, finds the corners of a rectangle within it and alligns it with the camera plane. Computer vision techniques are used to compare the recorded image with a prepared database of book covers to find the best match. The depth image is taken with a Microsoft Kinect camera and the application is coded in the C++ language using the OpenCV library. The plane detection uses the RANSAC algorithm while the corner detection algorithm relies on finding the most distant points and checking the corners between them. PCA projection and SIFT feature frequency histograms are used to describe the images classification is performed using SVM and *k*-NN algorithms and the euclidean distance to the best example of each class.

Keywords: computer vision, image recognition, range images

Poglavje 1 Uvod

Brezdotični vmesniki nam odpirajo možnosti bolj naravnega nadzora nad tehnološkimi rešitvami. Namesto vpisovanja podatkov in učenja kompleksnih nadzornih shem lahko z vsakdanjimi gibi sporočimo računalniku, kaj si želimo. Tako lahko tehnologijo, ki nam olajša različne aktivnosti, vključimo v naš vsakdan na intuitiven način. Uporaba različnih storitev za nas postane bolj udobna, za ponudnika pa je avtomatizacija cenejša in bolj učinkovita.

V primeru knjižnice moramo sistemu sporočiti, katero knjigo bi si radi izposodili oziroma jo vrnili. Najbolj naraven način sporočanja te informacije je, da knjigo pokažemo pred seboj. V pametni knjižnici je možno postaviti sistem, ki bo s tehnikami računalniškega vida v sliki prepoznal knjigo, ki jo uporabnik drži v roki. Za to ne potrebujemo nalepk in bralnikov črtnih kod, temveč lahko preprosto pogledamo platnico. Računalniku predstavimo enake informacije, kot jih opazujemo mi, ta pa poskrbi za njihovo uporabo v kompleksnem informacijskem sistemu.

Sistem mora v sliki prepoznati, kdaj uporabnik kameri kaže knjigo in kje se knjiga nahaja ter kako je obrnjena, nato pa lahko prepozna podobo na platnici. Vse to nam omogoči barvno-globinska kamera, ki ne izmeri le nagnjenih slik predmetov, ampak tudi njihovo obliko v prostoru. Z analizo te oblike lahko poravnamo objekt v sliki in ga primerjamo s podatkovno bazo. Med vsemi knjigami v zbirki prepoznamo pravo in jo ponudimo bralcu.

Diplomsko delo opisuje implementacijo takega sistema. Jedro besedila je razdeljeno na pet poglavij od drugega do šestega. V drugem poglavju se bomo spoznali s kamero Microsoft Kinect in preučili zajem slike. V tretjem bomo v zajeti sliki v več korakih prepoznali in popravili obliko knjige. V četrtem delu se bomo posvetili prepoznavanju zaznane knjige. Poglavje je razdeljeno na dva dela, iskanje značilk in njihovo klasifikacijo. V petem poglavju bomo pregledali uporabniški vmesnik in uporabljeno programsko opremo. Nazadnje bomo sistem testirali. Na zbirki knjižnih platnic bomo ovrednotili vse predstavljene pristope za klasifikacijo po natančnosti in hitrosti.

Poglavje 2 Zajem slike

Prvi korak do prepoznavanja knjižnih platnic je, da s kamero zajamemo njihove slike. V ta namen bomo uporabili bavno-globinsko kamero Microsoft Kinect. Spoznati se moramo z delovanjem in lastnostmi te kamere in s podatki, ki nam jih bosta oba senzorja posredovala. Vedeti moramo, kakšne informacije o globini nam senzor poda ter v katerih pogojih meritev ni mogoča. Predvsem je pri zajemu slike bistveno, da poskrbimo za ujemanje globinske in barvne slike. Opisali bomo tudi podatkovne strukture, uporabljene v programu in katere dele od njih smemo izpustiti za hitrejšo nadaljnjo obdelavo.

2.1 Microsoft Kinect

Slika 2.1: Deli senzorja Microsoft Kinect. (Avtor: Evan-Amos. Slika je v javni lasti.)

Microsoft Kinect (Slika 2.1) je vnosna naprava, ki s pomočjo infrardečega projektorja in kamere omogoča zajem trodimenzionalnega videa. Na trg je prišla leta 2010 kot prostoročni

vmesnik za konzolo Xbox 360, s pomočjo katerega igralec z lastnimi gibi neposredno nadzoruje like v igri. Leta 2012 je bila izdana enačica za osebni računalnik, ki se od prejšnje razlikuje predvsem po programski opremi. Zaradi inovativne funkcionalnosti in nizke cene je Kinect po izidu prosto dostopnih razvijalskih orodij postal popularno orodje za različne aplikacije računalniškega vida, ki zahtevajo trodimenzionalne podatke in predvsem prepoznavanje uporabnikove poze. Z izidom konzole Xbox One ga je nadomestil novejši Kinect, ki uporablja drugačno tehnologijo za merjenje globine.

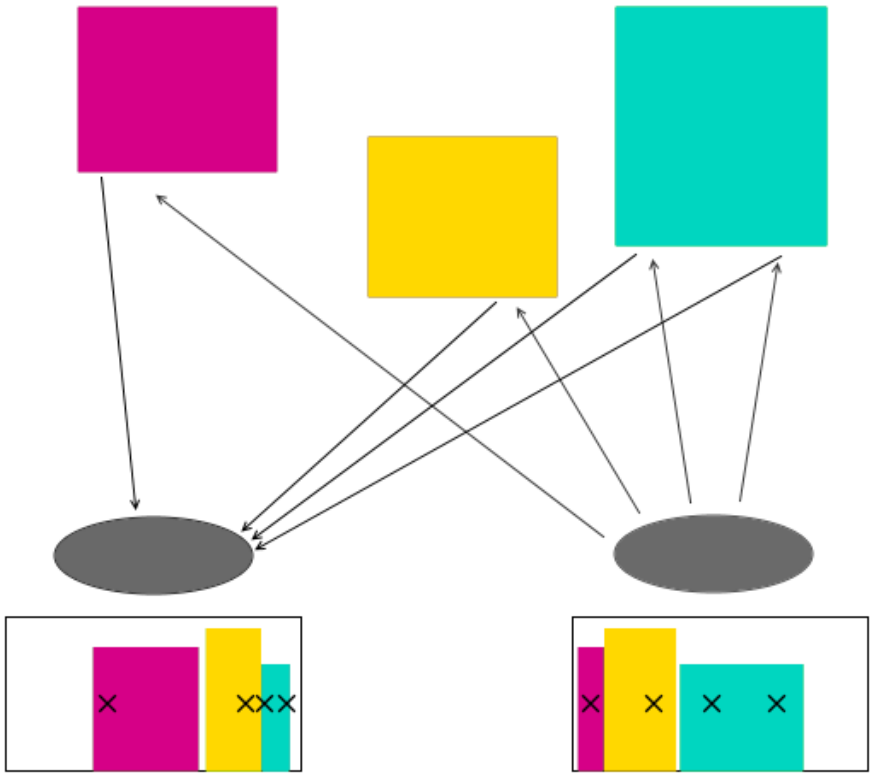
2.1.1 Kamera RGB

Barvna kamera je v prvi vrsti namenjena zajemu slike uporabnika. Kamera lahko sprejema video ločljivosti 640 x 480 slikovnih elementov s hitrostjo 30 okvirjev na sekundo ali 1280 x 1024 slikovnih elementov s hitrostjo približno 10 okvirjev na sekundo. Barvna globina slike je 24 bitov, torej 8 bitov za kanale rdeče, zelene in modre barve, kar predstavlja 16 milijonov različnih odtenkov [*16*]. Vidno polje kamere obsega kot 57 stopinj v vodoravni smeri in 43 stopinj v navpični smeri [*14*].

2.1.2 Globinski senzor

Kinect uporablja infrardeči projektor in infrardečo kamero za zajem globinske slike. Kamera snema video ločljivosti 640 x 480 slikovnih elementov s hitrostjo 30 okvirjev na sekundo. Slika ima samo en kanal, ki ima globino 11 bitov, torej je teoretično možno meriti globino na 2048 vrednosti natančno [*16*]. 7,5 cm desno od infrardeče kamere se nahaja infrardeči projektor. Globinski senzor za konzolo Xbox 360, ki je bil uporabljen v diplomskem delu, je omejen na globine od 50 cm do 400 cm [*6*].

Kinectov globinski senzor za ocenjevanje globine uporablja tehniko, ki deluje po principu strukturirane svetlobe. Namesto da bi oddaljenost objektov izmerili na podlagi razlike med dvema slikama, posnetima iz različnih kotov, pri Kinectu vlogo druge kamere prevzame infrardeči projektor. V prostor projicira vzorec točk, ki je bil pripravljen vnaprej in ga zato ni treba meriti. Te točke so zaradi velike valovne dolžine svetlobe nevidne tako uporabniku kot barvni kameri, zazna pa jih infrardeča kamera. Slika 2.2 prikazuje, kako se vzorec, projiciran s projektorja na desni, zamakne v pogledu kamere na levi. Zamik vzorca je odvisen od kotov med projektorjem, točkami na površini in infrardečo kamero. Čim manjša je razdalja med napravo in površino, na kateri ležijo točke, tem večji je ta kot in bolj se slika razlikuje od predvidene [*23*].



Slika 2.2: Prikaz delovanja Kinectovega stereo vida.

2.1.3 Mikrofon in nagib

Kinect vključuje tudi motorizirano stojalo, ki lahko kamero zavrti za 43 stopinj v navpični smeri in 57 stopinj v vodoravni smeri in s tem sledi igralčevim premikom. Trenutno smer snemanja naprava izmeri s triosnim pospeškometrom [*14*].

Poleg tega je Kinect opremljen s štirimi mikrofoni, ki snemajo zvok s frekvenco vzorčenja 16 kHz in bitno globino 16 bitov. Ti mikrofoni se lahko uporabljajo za snemanje zvoka med igro ali video pogovorom in za glasovne ukaze igralni konzoli. Na podlagi razlik v sprejetem zvoku in znane medsebojne lokacije mikrofonov Kinect locira zvoke v prostoru in omeji smer snemanja v določene smeri. Poudarjanje zvoka iz smeri igralca se uporablja za filtriranje hrupa [*14*].

2.1.4 Gonilniki

Kinect je bil leta 2010 razvit za povezavo s konzolo Xbox360. Ta za povezavo uporablja vmesnik USB, zato je bil strojno kompatibilen tudi z osebnimi računalniki in pojavilo se je zanimanje za uporabo strojne opreme za druge namene. Prvi odprtokodni gonilniki so bili razviti za operacijski sistem Linux, potem ko je podjetje Adafruit za njih razpisalo nagrado.

Kasneje je Microsoftov partner v razvoju Kinecta, podjetje Primesense, izdalo svoje gonilnike NITE. Ti so bili vključeni v ogrodje OpenNI, ki je bilo zaradi lahke integracije s knjižnico OpenCV, uporabljeno v tem projektu.

2.2 Zajemanje slik

Generatorju barvnih slik moramo določiti ločljivost in frekvenco zajema slike. Na voljo imamo ločljivost 640 x 480 slikovnih elementov s frekvenco 30 Hz in ločljivost 1280 x 1024 slikovnih elementov, ki naj bi delovala pri frekvenci 15 Hz. Toda testi kažejo, da se dejanska hitrost giblje okoli 10 Hz [*16*]. Za potrebe programa je druga opcija bolj primerna. Ker bo uporabnik knjigo držal nekoliko stran od kamere in bo morda nagnjena, bomo izgubili precej podrobnosti, zato potrebujemo čim višjo ločljivost v začetnih podatkih. Generator globinskih slik ne omogoča višje ločljivosti in bo deloval pri ločljivosti 640 x 480 in frekvenci 30 Hz. V programu sprožimo procesiranje šele, ko dobimo podatke iz obeh virov.

Slika 2.3: Prekrivanje slike RGB in globinske slike. Levo: Sliki brez registracije. Desno: Sliki z registracijo.

Pomembno je, da globinsko in barvno sliko registriramo. Registracija je proces, pri katerem poravnamo dve ali več slik, ki vsebujejo isto sceno, a se ne ujemajo popolnoma. Pri Kinectu ima infrardeča kamera nekoliko drugačne optične značilnosti in leži 2,5 cm levo od barvne, zato je med njunima slikama majhna razlika v perspektivi. Globinska slika je v primerjavi z barvno zamaknjena, obrezana in tudi nekoliko deformirana (Slika 2.3). Točno preslikavo, ki bi upoštevala vse te faktorje, bi težko ocenili sami, zato vključimo možnost v gonilnikih OpenNI. Po registraciji se v vseh tabelah iz generatorja globinskih slik nahajajo podatki za pripadajoče točke v barvni sliki. Ko je naprava odprta za komunikacijo, lahko začnemo

sprejemati okvirje za obdelavo. Okvirje sprejmemo v obliki treh tabel: slika BGR, maska veljavne globine in oblak točk.

Maska veljavne globine vsebuje binarne vrednosti za vsako točko, ki povedo, ali je meritev v tej točki pravilna ali ne. Kinectov globinski senzor namreč ne more izmeriti predmetov, ki se nahajajo preblizu ali predaleč od kamere. Na sliki vidimo tudi sence okoli bližnjih predmetov. Če se med površino, na katero sveti projektor, in infrardečo kamero nahaja drug predmet, namreč ta piko zakrije kameri. Hkrati pa kamera vidi del področja za predmetom, ki prestreza žarek. Ker predmet blokira žarke, pik za njim ni, čeprav je področje kameri vidno. Na takšnih lokacijah je meritev nemogoča. Sistem je manj zanesljiv tudi na odsevnih površinah. Končno pa se v tabeli pojavijo področja, ki jih infrardeča kamera zaradi ožjega kota gledanja ne vidi, a so bila vključena med registracijo, saj jih vidi kamera RGB. V vseh teh primerih maska veljavne globine vsebuje vrednost 0, v ostalih pa 1. Slika BGR je tridimenzionalna tabela, ki vsebuje barvo vsake točke v okvirju kot moč modre, zelene in rdeče svetlobe (v tem vrstnem redu). Vsaka barvna komponenta vsake točke je v tabeli zapisana kot celo število med 0 in 255. Globinsko sliko pa lahko namesto tabele globin za vsako točko sprejmemo kot oblak točk, v katerem so že izračunane lokacije vseh površin v prostoru. Oblak točk je prav tako trodimenzionalna tabela, ki vsebuje izmerjeno lokacijo vsake točke kot koordinate x, y in z v metrih relativno na kamero. Vsaka komponenta vsake točke je v tabeli zapisana kot 32-bitno decimalno število v zapisu plavajoče vejice. Za vsak okvir v nestisnjeni obliki sprejmemo približno 7,3 MB podatkov.

Za bolj gladko delovanje programa je potrebno, da se ne ukvarjamo po nepotrebnem z okvirji, v katerih knjige ni. Predpostavili bomo, da se mora za uporabo aplikacije knjiga nahajati med 50 cm in 100 cm od kamere in da v tem območju ni ničesar razen dela uporabnikove roke, ki drži knjigo. Na začetku preverimo mrežo 10 x 10 točk, ki so enakomerno razporejene po sliki. Knjiga, ki jo uporabnik kaže kameri, bo nedvomno večja od 64 x 48 slikovnih elementov, torej bo pokrivala vsaj eno od teh točk. Ko v vsaj eni izmed točk izmerimo oddaljenost do 100 cm, sprožimo zaznavanje knjižnih platnic.

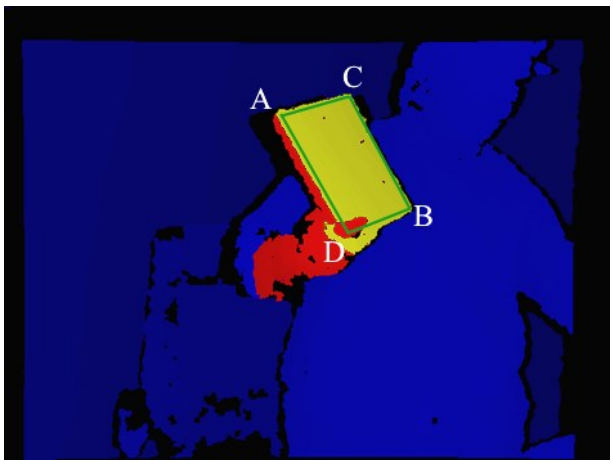
Poglavje 3 Zaznavanje knjižnih platnic

3.1 Pregled postopka

Vsak okvir postopoma omejimo na željeno območje (Slika 3.1). Ker je knjiga ravninski predmet, iz tega območja lahko izračunamo projekcijo v sliko, kot bi bila posneta pod pravim kotom in problem iz prepoznavanja trodimenzionalnega predmeta poenostavimo v prepoznavanje dvodimenzionalne slike.

Potrebni koraki so sledeči:

1. Omejimo področje po razdalji od kamere.
2. Zmanjšamo ločljivost relevantnega dela za hitrejšo obdelavo.
3. V obrezanem delu prepoznamo ravnino in odstranimo moteče elemente (roko).
4. Poiščemo štiri oglišča.
    4.1. A je najbolj oddaljena točko od središča platnice.
    4.2. B je najbolj oddaljena točka od A.
    4.3. C in D sta najbolj oddaljeni točki pravokotno na daljico AB.
    4.4. D zaradi napak v vhodnih podatkih popravimo na podlagi A, B in C.
5. Izrežemo in naravnamo sliko s preslikavo iz najdenih štirih oglišč v standardna.
6. Normaliziramo svetlost knjige.



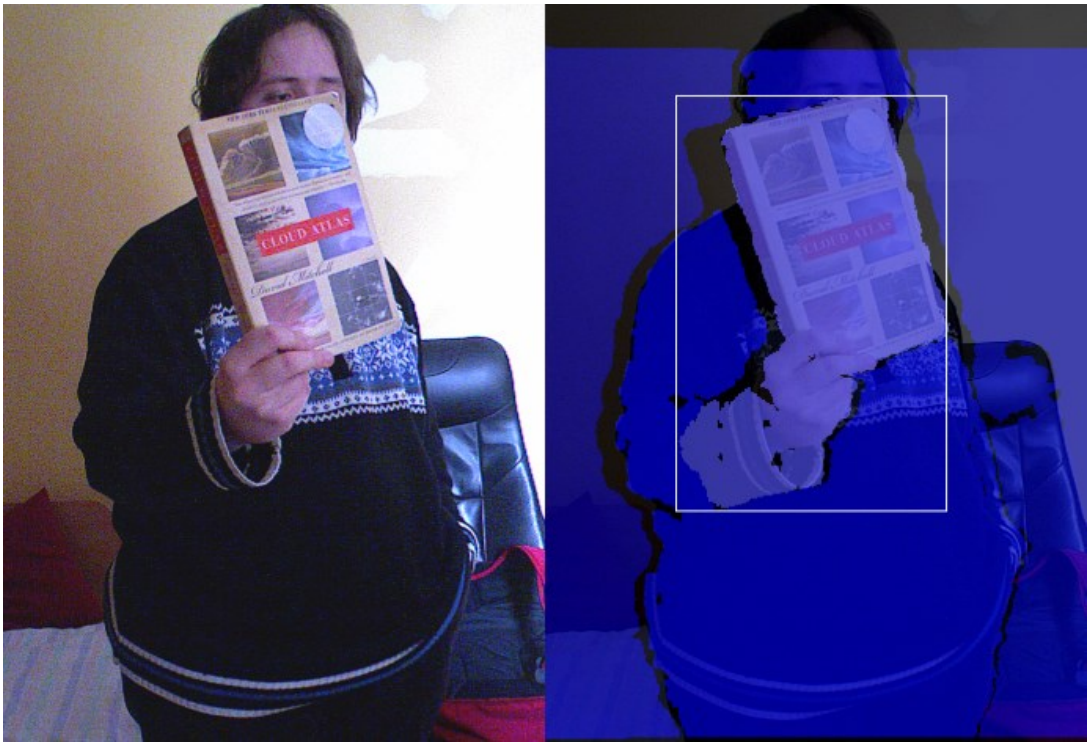
Slika 3.1: Globinska slika z označenimi regijami. Črna področja so napake v meritvah, modra so predaleč od kamere, rdeča so bila odstranjena z algoritmom RANSAC, rumena je najdena ravnina.

V tem poglavju si bomo podrobno ogledali celoten postopek s poudarkom na problemih učinkovitega pregledovanja globinske slike in prepoznavanju pravokotnika v nenatančni meritvi in nagnjeni perspektivi. Pogledali bomo tudi izračun perspektivne preslikave v standardna oglišča in normalizacijo svetlosti.

3.2 Zaznavanje ravnine

Za hitrejše zaznavanje knjižnih platnic je dobro, da že vnaprej obrežemo globinsko sliko na vodoraven pravokotnik okoli knjige, čeprav ta ne ustreza njeni obliki. Tudi polna ločljivost ni potrebna, zato jo zmanjšamo. V globinski sliki poiščemo najbolj levo, desno, zgornjo in spodnjo točko, ki ima primerno razdaljo do kamere. Izmerimo, katera stranica pravokotnika je krajša in pravokotnik zmanjšamo tako, da bo krajša stranica dolga 50 slikovnih elementov (Slika 3.2).

Pred iskanjem oglišč moramo knjigo čimbolj natančno izločiti iz okvirja. V področju, v katerem bomo iskali oglišča, se ne sme nahajati ozadje knjige, poleg tega ne smemo odrezati večjega dela knjige. Iz slike ne moremo izbrisati prstov, ki prekrivajo del platnice, želimo pa odrezati roko, ki drži knjigo. Za začetek torej v globinski sliki prepoznamo ravnino, na kateri leži knjiga in odstranimo vse točke, ki se ne nahajajo na njej.

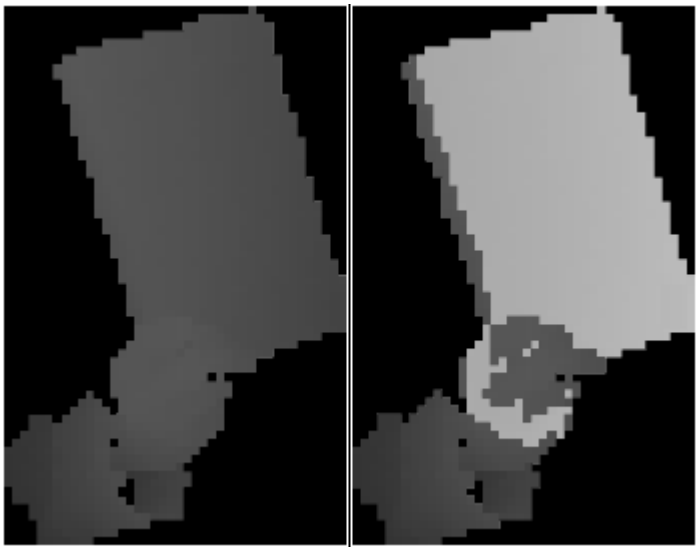


Slika 3.2: Omejevanje območja po globini in obrez.

Iskanje ravnine izvedemo z algoritmom RANSAC (Algoritem 1). Namenjen je iskanju matematičnega modela v naboru podatkov, ki vsebuje odstopanja. Z naključnim vzorčenjem izberemo najmanjše število vzorcev, ki je potrebno za definiranje modela in nato preverimo, kako dobro najdeni model opisuje celoten nabor podatkov. Določimo dovoljeno razdaljo od ravnine in najnižje zahtevano število ustreznih točk. Za vsak vzorec izračunamo oceno odstopanja. Če se znotraj definirane meje ne nahaja dovolj točk, postopek ponovimo od začetka. Ker algoritem temelji na naključju, ne zagotavlja odkritja pravilnega rezultata. Definirati moramo maksimalno število poskusov, po katerem opustimo procesiranje trenutne slike v primeru, da model ni bil najden. Vendar pa je verjetnost uspeha zelo visoka ob predpostavki, da je v naboru podatkov precej manj motečih točk kot tistih, ki ustrezajo željenemu matematičnemu modelu [*20*].

Naš model je ravnina in za njen opis potrebujemo tri točke, ki ne ležijo na isti premici. Vektorski produkt med vektorjema od prve točke $\overrightarrow{T_1}$ do druge $\overrightarrow{T_2}$ in tretje $\overrightarrow{T_3}$ je pravokoten na oba in predstavlja normalo ravnine $\vec{n} = (\overrightarrow{T_2} - \overrightarrow{T_1}) \cdot (\overrightarrow{T_3} - \overrightarrow{T_1})$, v kateri ležijo. Prvo izmed izbranih točk določimo kot izhodišče ravnine in vse točke, ki so od nje odmaknjene za vektor, pravokoten na normalo, se nahajajo v ravnini. Za oceno napake uporabimo razdaljo med dano točko $\vec{T}$ in ravnino. To dobimo tako, da vektor od izhodišča do dane točke projiciramo na normalo in izračunamo dolžino projekcije [*21*]

$$d = \frac{\left|(\vec{T} - \overrightarrow{T_1}) \cdot \vec{n}\right|}{|\vec{n} \cdot \vec{n}|}.$$



Slika 3.3: Detektirana ravnina.

Ko najdemo ravnino, iz globinske slike odstranimo točke, ki niso dovolj blizu nje (Slika 3.3). Tudi če platnico zaznamo zelo natančno, je v sliki še vedno motnja okoli oglišča, za katerega uporabnik drži knjigo. Te motnje ne moremo popraviti, a pri iskanju pravokotnika v ravnini nam bo v pomoč, če bomo vedeli, v katerem izmed oglišč se nahaja. Zato dvakrat izračunamo težišče kot povprečni x in y koordinati točk: prvič upoštevamo knjigo skupaj z roko, drugič samo knjigo po odstranitvi roke. Težišče se bo premaknilo v nasprotno smer od področja, s katerega je bila odstranjena roka.

```
RazdaljaDoRavnine
Vhod: (točka, center, normala)
      dolžinaNormale <- Koren (SkalarniProdukt( normala, normala))
      vrni abs( SkalarniProdukt( normala, točka – center )) / dolžinaNormale

Ransac
Vhod: (maskaVeljavnihGlobin, oblakTočk, maksIteracij, maksRazdalja, minRezultat)
      indeksi <- []
      h <- oblakTočk.višina
      w <- oblakTočk.širina
      za vsak x, 0 <= x < w
            za vsak y, 0 <= y < h
                  če maskaVeljavnihGlobin[ x, y ] == 1
                        indeksi.vstavi( y * w  + x )
      štIndeksov <- indeksi.dolžina
      če štIndeksov < minRezultat
            vrni 0
      najŠtTočk <- 0
      najCenter <- [0,0,0]
      najNormala <- [0,0,0]
      za vsak i, 0 <= i < maksIteracij
            indeks <- indeksi[ naključno(0, štIndeksov) ]
            c <- oblakTočk[ indeks /  w, indeks % w ]
            indeks <- indeksi[ naključno(0, štIndeksov) ]
            b <- oblakTočk[ indeks /  w, indeks % w ]
            indeks <- indeksi[ naključno(0, štIndeksov) ]
            a <- oblakTočk[ indeks /  w, indeks % w ]
            če b == a ali c == a ali b == c
                  preskoči obrat zanke
            normala <- skalarniProdukt( a – c, b – c );
            štTočk <- 0
            za vsak j, 0 <= j < štIndeksov
                  p <- oblakTočk [ indeksi[ j ] / w, indeksi[ j ] % w ]
                  razdalja <- RazdaljaDoRavnine( p, c, normala )
                  če razdalja <= maksRazdalja
                        štTočk <- štTočk + 1
            če štTočk > najŠtTočk
                  najŠtTočk <- štTočk
                  najNormala <- normala
                  najCenter <- c
            če štTočk > minRezultat
                  prekini zanko
```

```
      če najŠtTočk > minRezultat
            za vsak i, 0 <= i < štIndeksov
                  indeks < indeksi[ i ]
                  p = oblakTočk[ indeks / w, indeks % w ]
                  razdalja <- RazdaljaDoRavnine(p, najCenter, najNormala)
                  če razdalja > maksRazdalja
                        maskaVeljavnihDolžin[ indeksi[ i ]] <- 0
            vrni 1
      sicer
            vrni 0
```

Algoritem 1: RANSAC.

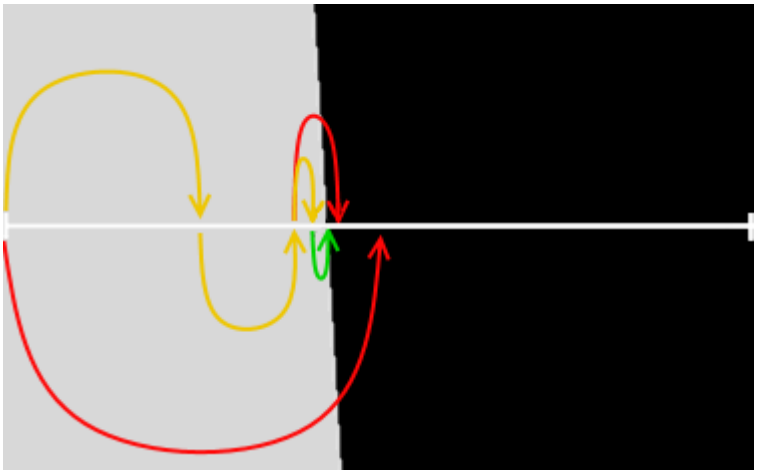
Načeloma je za izrezano knjižno platnico dovolj 200 iteracij algoritma RANSAC z zahtevo, da najdemo ravnino z najmanj 800 točkami, ki se nahajajo do 0,5 cm od nje. Če slika ne vsebuje vsaj 800 točk, verjetno ni primerna in že na tem mestu zavržemo okvir. Če želimo večjo natančnost, zahtevamo, da se vedno izvede polnih 200 iteracij in se izbere ravnina, ki vsebuje največ točk. Če želimo večjo hitrost, algoritem zaključimo ob prvi najdeni ravnini z najmanj 800 primernimi točkami. Najpomembnejša optimizacija algoritma je vnaprej pripravljen seznam sprejemljivih indeksov za naključno vzorčenje. Na sliki, obrezani v navpičen pravokotnik, je veliko praznega prostora, če je knjiga obrnjena poševno. Še več ga je okoli dela uporabnikove roke, če se velik del nje nahaja v območju do 100 cm od kamere. Ker v algoritmu RANSAC mnogokrat preiščemo celotno sliko, je torej zelo koristno, da že prej v enem obhodu izločimo vse točke, ki ne ustrezajo osnovnim kriterijem in se izognemo nepotrebnemu preverjanju. Točke, ki so bile pravilno izmerjene in se nahajajo od 50 cm do 100 cm od kamere, shranimo v vektor, iz katerega bomo naključno izbirali vzorce, ostale pa odstranimo iz tabele. S to optimizacijo bistveno zmanjšamo število ponovitev algoritma. Ko poznamo točke v ravnini, lahko med njimi poiščemo oglišča.

3.3 Iskanje oglišč v ravnini

Za nadaljnjo obdelavo moramo najti oglišča knjige, da jo potem naravnamo v navpičen pravokotnik. Problem predstavlja ureditev podatkov, saj točke v oblaku niso urejene po njihovi poziciji v tridimenzionalnem prostoru, temveč po poziciji, kamor se preslikajo v dvodimenzionalni ravnini kamere. Dostopamo torej lahko do katerekoli točke v sliki in najdemo njene prostorske koordinate, ne moremo pa za poljubno trodimenzionalno pozicijo izračunati, kam v sliki se projicira. To nam preprečuje zamik kamer, ki smo ga med zajemom slike rešili z registracijo, a ker je transformacija vgrajena v gonilnike OpenNI in vključuje deformacijo slike, ne poznamo njenega inverza. Oglišča nagnjenega pravokotnika moramo torej najti v njegovi dvodimenzionalni projekciji. Pri preiskovanju se ne moremo zanesti na prave kote v obliki podatkov v tabeli. Prav tako niso zanesljive ravne stranice, saj globinska

slika ni dovolj natančna in vsebuje motnje, nazobčane robove in nepredvidljivo obliko uporabnikove roke. Lahko pa se zanesemo na razdalje. Vsa štiri oglišča bodo vedno ležala daleč od težišča in drugo od drugega.

Postopek iskanja začnemo tako, da najdemo točko A, ki je najbolj oddaljena od sredine. Nato najdemo točko B, ki je najbolj oddaljena od točke A. Našli smo diagonalo, ki ne glede na orientacijo razdeli pravokotnik na dva trikotnika. Preostali oglišči C in D sta najbolj oddaljeni točki od diagonale AB.

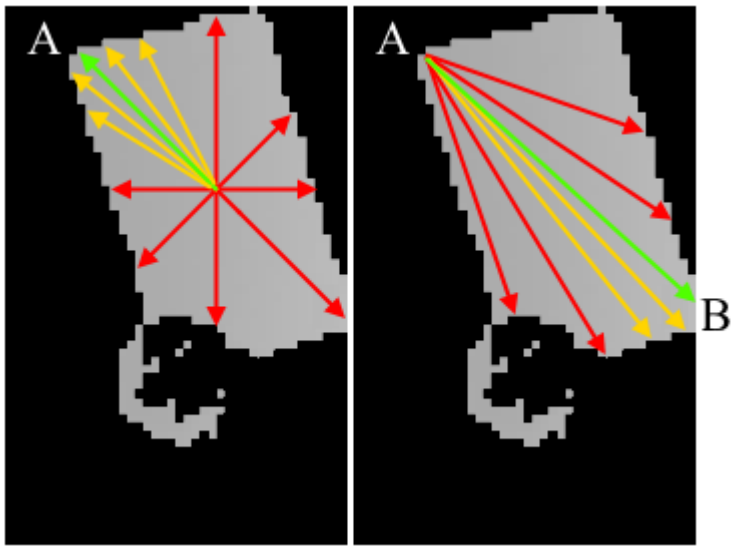


Slika 3.4: Iskanje najbolj oddaljene točke v ravnini v določeni smeri.

Pri preiskovanju ravnine se želimo izogniti nepotrebnemu preverjanju vseh točk. Za iskanje najbolj oddaljene točke na dani daljici uporabimo pristop, podoben bisekciji (Algoritem 2), le da se izogibamo preseganju rezultata (Slika 3.4). Iz težišča, ki smo ga izračunali po algoritmu RANSAC, si izberemo smer in razdaljo d in preverimo, ali se lokacija, ki je za $d/2$ oddaljena od izhodišča, nahaja znotraj ravnine. Če se, se premaknemo tja. Sedaj preverimo, ali se lokacija za $d/4$ v isto smer nahaja znotraj ravnine in če se, skočimo tja. Število preverjanj določimo vnaprej, pri čemer štejemo tudi tista preverjanja, po katerih se nismo premaknili. Tega postopka ne moremo zaključiti pred iztekom števila korakov. Pri bisekciji se v vsakem koraku odločimo, na kateri strani leži željeni rezultat in se premaknemo proti njemu z vedno krajšimi skoki. Toda v našem problemu se točke, ki so daleč od roba ravnine, ne ločijo od točke, ki leži točno na robu. Da bi vsakič preverili, ali točka leži na robu, bi morali preveriti naslednjo točko ob njej, s čimer bi podvojili trajanje funkcije. Pomembno je tudi, da ne moremo zaključiti v praznem polju, saj tam globina ni definirana. Parameter d moramo nastaviti dovolj visoko, da je izhodišče od vseh štirih oglišč slike (in ne knjige v sliki) oddaljeno kvečjemu d. Tako najdemo najbolj oddaljeno točko od središča v izbrani smeri.

```
PreiščiDaljico
Vhod: (maskaVeljavnihGlobin, začetniX, začetniY, kot, dolžinaDaljice,
štKorakov)
      dolžina <- dolžinaDaljice / 2
      x <- začetniX
      y <- začetniY
      najDolžina <- 0
      za vsak i, 0 <= i < štKorakov
            tempX <- x + kosinus( kot ) * dolžina
            tempY <- y + sinus( kot ) * dolžina
            če 0 <= tempX < maskaVeljavnihGlobin.širina
            in če 0 <= tempY < maskaVeljavnihGlobin.višina
            in če maskaVeljavnihGlobin[ x, y ] == 1
                  x <- tempX
                  y <- tempY
                  najDolžina <- najDolžina + dolžina
            dolžina <- dolžina / 2
      vrni najDolžina
```

Algoritem 2: Iskanje najbolj oddaljene točke v ravnini v določeni smeri.



Slika 3.5: Iskanje prvih dveh oglišč z deljenjem kotov.

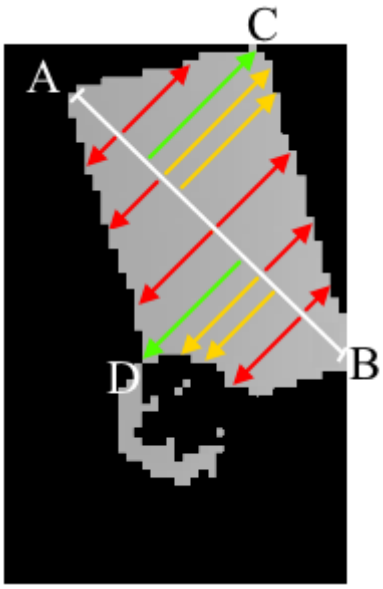
To funkcijo uporabimo v naslednjem koraku. Pregledovali bomo daljice v različnih smereh od centra in poskušali najti smer, v kateri se nahaja najbolj oddaljeno oglišče (Slika 3.5)(Algoritem 3). Kota 360 stopinj ni smiselno deliti na dvoje, saj namesto linearnega naraščanja zdaj razdaljo, ki jo bomo našli v določeni smeri, določa nepredvidljiva oblika nagnjenega štirikotnika v globinski sliki. V prvi fazi kot razdelimo na večje število manjših kotov in v vsaki smeri poiščemo najbolj oddaljeno točko. V drugi fazi še enkrat pregledamo manjši kot okoli najdene smeri in ga razdelimo na nekaj še manjših. Na ta način najdemo

točko A. Za točko B uporabimo isti pristop, a nam ni treba preiskati kota 360 stopinj okoli točke A, saj vemo, da je točka A oglišče in ima na nasprotni strani od središča prazen prostor, zato se omejimo na področje v smeri središča.

```
PreiščiKot
Vhod: (maskaVeljavnihGlobin, začetniX, začetniY, začetniKot, končniKot,
dolžinaDaljice, dKorakov, štKorakovVFazi, faza)
      štKorakov <- korakiVFazi[ faza ]
      najKot <- 0
      najDolžina <- 0
      deltaKot <- (končniKot - začetniKot) / štKorakov
      za vsak i, 0 <= i < štKorakov
            kot <- začetniKot + deltaKot * i
            dolžina <- PreiščiDaljico( maskaVeljavnihGlobin, začetniX, začetniY,
                                       kot, dKorakov
            če dolžina > najDolžina
                  najDolžina <- dolžina
                  najKot <- kot
      če faza < štKorakovVFazi.dolžina
                  najKot <- PreiščiKot( začetniX, začetniY, najKot - deltaKot,

                        najKot + deltaKot, dolžinaDaljice, dKorakov,
                        štKorakovVFazi, faza + 1)
      če najRazdalja > 0
            vrni najKot
      sicer
            vrni -1
```

Algoritem 3: Iskanje smeri v kotu z najbolj oddaljeno točko v ravnini.



Slika 3.6: Iskanje drugih dveh oglišč z deljenjem diagonale.

Za iskanje točk C in D dvakrat uporabimo podoven pristop, a namesto kota okoli centra preiskuje daljico med točkama A in B (Algoritem 4). Na njej iščemo lego, kjer se pravokotno na daljico nahajata najbolj oddaljeni točki (Slika 3.6). V prvem koraku spet preizkusimo več različnih leg, nato pa natančneje pregledamo območje okoli najboljšega rezultata.

```
PreiščiVrsto
Vhod: (maskaVeljavnihGlobin, začetniX, začetniY, začetnaPozicija, končnaPozicija,
kot, dolžinaDaljice, dKorakov, štKorakovVFazi, faza)
      štKorakov <- korakiVFazi[ faza ]
      najPozicija <- 0
      najDolžina <- 0
      delta <- (končnaPozicija - začetnaPozicija )
      za vsak i, 0 <= i < štKorakov
            pozicija <- začetnaPozicija + delta * i / štKorakov
            trenutniX <- začetniX + kosinus( kot ) * pozicija
            trenutniY <- začetniY + sinus( kot ) * pozicija
            dolžina <- PreiščiDaljico( maskaVeljavnihGlobin, trenutniX,
                        trenutniY, kot + PI, dolžinaDaljice, dKorakov)
            če dolžina > najDolžina
                  najDolžina <- dolžina
                  najPozicija <- pozicija
      če faza < štKorakovVFazi.dolžina
            najPozicija <- PreiščiVrsto( začetniX, začetniY, najPozicija - delta,
                        najPozicija + delta, kot, dolžinaDaljice,  dKorakov,
                        štKorakovVFazi, faza + 1)
      če najDolžina > 0
            vrni najPozicija
      sicer
            vrni -1
```

Algoritem 4: Iskanje najbolj oddaljene točke v ravnini pravokotno na daljico.

Ker se v sliki pogosto pojavi motnja v obliki manjkajočih ali odvečnih točk okoli uporabnikove dlani (Slika 3.6), je verjetno oglišče, ki se tam nahaja, ocenjeno manj natančno ali povsem napačno. Za identifikacijo oglišča uporabimo prej izračunano smer premika težišča slike po odstranitvi roke iz slike. Preverimo vse točke (x, y) in izračunamo, pod kakšnim kotom φ so odmaknjene od težišča (x_{avg}, y_{avg}):

$$\varphi = \begin{cases} \pi * \left(1 + 0.5 * sig\left(y - y_{avg}\right)\right), & x = x_{avg} \\ tan^{-1}\left(\frac{y - y_{avg}}{x - x_{avg}}\right) + \pi, & x < x_{avg} \\ tan^{-1}\left(\frac{y - y_{avg}}{x - x_{avg}}\right) + 2\pi, & x > x_{avg} \wedge y > y_{avg} \\ tan^{-1}\left(\frac{y - y_{avg}}{x - x_{avg}}\right), & x < x_{avg} \wedge y > y_{avg} \end{cases}.$$

Nato izračunamo razliko

$$\Delta\varphi = \begin{cases} |\varphi - \varphi_{premik}|, & |\varphi - \varphi_{premik}| < \pi \\ 2 * \pi - |\varphi - \varphi_{premik}|, & |\varphi - \varphi_{premik}| \geq \pi \end{cases}$$

med tem kotom in kotom φ_{premik}, pod katerim je novo težišče odmaknjeno od starega. Točka, ki ima najmanjšo razliko $\Delta\varphi$, se nahaja v prizadetem predelu slike.

Poglejmo si primer, ko uporabnik drži knjigo za oglišče D. Poznamo koordinate točk A, B in C v ravninski sliki, torej lahko v oblaku točk dostopamo tudi do njihovih koordinat v prostoru. Točka B leži diagonalno na drugi strani pravokotnika, točki A in C pa predstavljata ostali dve oglišči. Če prostorske koordinate točke B projiciramo čez daljico med A in C, dobimo idealne prostorske koordinate točke D: $\overrightarrow{D_{idealna}} = \vec{A} + \vec{C} - \vec{B}$. Sedaj še enkrat preiščemo dvodimenzionalno sliko in najdemo točko v ravnini, katere prostorske koordinate so najbližje idealni točki.

Ker je bilo četrto oglišče zaradi uporabnikove roke lahko ocenjeno povsem narobe, ga nima smisla uporabiti kot izhodišče za optimizacijo. Za izhodišče si spet izberemo center slike in v različnih smereh iščemo najbližjo točko (Algoritem 5). Preiskujemo kot 90 stopinj okoli smeri, nasprotne točki B. Tokrat nekoliko težje sklepamo o urejenosti podatkov. Ker je področje iskanja kvečjemu četrtina slike, lahko opustimo pristop »deli in vladaj« in v enem koraku iščemo z višjo natančnostjo.

```
najdiPraviKot
Vhod: (maskaVeljavnihGlobin, oblakTočk, začetniX, začetniY, začetniKot, končniKot,
dolžinaDaljice, štKorakovK, štKorakovD, tA, tB, tC)
      tD <- tA + tC - tB
      najTočka <- (0, 0, 0)
      najBližina <- FLOAT_MAX
      za vsak i, 0 <= i < štKorakovK
            kot <- začetniKot + (končniKot - začetniKot) / štKorakovK * i
            za vsak j, 0 <= j < štKorakovD
                  razdalja <- dolžinaDaljice * j / štKorakovD
                  Dx <- začetniX + kosinus( kot ) * razdalja
                  Dy <- začetniY + sinus( kot ) * razdalja
                  če maskaVeljavnihGlobin [ Dx, Dy ] == 1
                        tX <- oblakTočk[ Dx, Dy ]
                        bližina <- koren( skalarniProdukt( tX - tD, tX - tD )
                        če bližina < najBližina
                              najTočka <- tX
                              najBližina <- bližina
      vrni najTočka
```

Algoritem 5: Iskanje točke, najbližje idealnemu pravemu kotu.

Za konec še enkrat preverimo kvaliteto najdenih točk. Preverimo dva kriterija. Postavimo mejo za najkrajšo možno stranico knjige, saj se s tem izognemo delnim rezultatom zaradi nenatančno detektirane ravnine. Predvsem pa je bistveno, da preverimo pravilnost kotov. Če okvir vsebuje kot, ki je preveč oddaljen od devetdeset stopinj, ga zavržemo, saj sicer pri transformaciji pride do pretiranega striženja. Kot $\varphi_{\sphericalangle ABC}$ izračunamo po enačbi [*21*]:

$$\varphi_{\sphericalangle ABC} = cos^{-1}\left(\frac{(\vec{A}-\vec{B})\cdot(\vec{C}-\vec{B})}{\sqrt{((\vec{A}-\vec{B})\cdot(\vec{A}-\vec{B}))*((\vec{C}-\vec{B})\cdot(\vec{C}-\vec{B}))}}\right).$$

Z najdenimi oglišči lahko nadaljujemo s preslikavo knjižne platnice v idealno obliko za nadaljnno obdelavo.

3.4 Rektifikacija slike

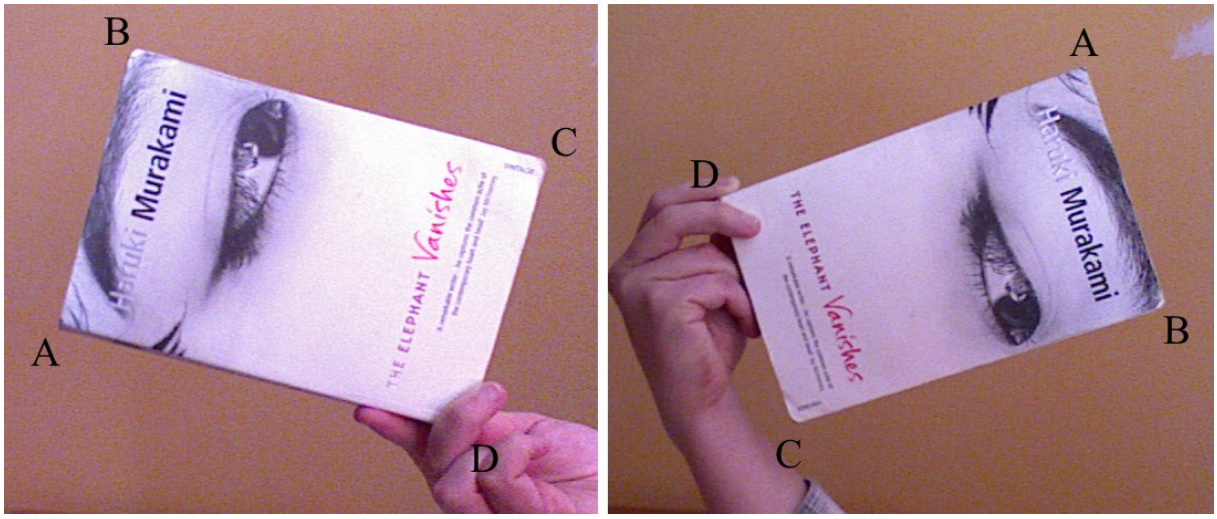
Slika 3.7: Naravnana in obrezana slika knjige.

Vse knjige bomo preslikali v navpičen pravokotnik (Slika 3.7). Maksimalne dimenzije tega pravokotnika lahko ocenimo grafično, tako da večjo knjigo pokažemo kameri z razdalje okoli 50 cm in jo nato v posnetku izrežemo in izmerimo njene stranice. Za pravokotnik lahko izberemo razmerje stranic, podobno A4-formatu papirja. Označimo dimenzije pravokotnika

kot w (širina) in h (višina). Potrebna je preslikava med dvema skupinama štirih točk, torej v najdene štiri točke v sliki preslikamo točke (0,0), (0,h), (w,h), (w,0). Pomembno je, da za vsako izmed štirih točk vemo, kateri od drugih štirih ustreza. Izračunati moramo, kako je knjiga obrnjena, si oglišča zapisati v smeri urnega kazalca in vsakič začeti pri istem oglišču, sicer bomo sliko obrnili v napačno smer oziroma bo prišlo do nenavadnih deformacij, če zamešamo vrstni red.

Zahtevati moramo, da uporabnik po ravnini, vzporedni s kamero, knjige ne zavrti za več kot 90 stopinj, saj sicer ne bi bilo jasno, ali gre za knjigo, obrnjeno za 120 stopinj v levo ali za 60 stopinj v desno. Če bi se program odločil narobe, bi bila knjiga obrnjena za 180 stopinj.

Pri iskanju prvega oglišča se zaradi možne rotacije knjige ne moremo opreti na bližino oglišča do zgornjega levega oglišča slike. Če je knjiga obrnjena 80 stopinj v levo, mu bo zgornje desno oglišče bližje. Če je obrnjena 80 stopinj v desno, pa mu bo spodnje levo oglišče bližje. V knjigi navpičnega formata bo stranica v smeri urnega kazalca od prvega oglišča krajša od stranice v nasprotni smeri. Torej vrstni red oglišč zamaknemo za eno oglišče, če podatki ne ustrezajo temu kriteriju. Poleg tega se zgornje levo oglišče, če nagib knjige ne presega kota 90 stopinj, nikoli ne bo nahajalo nižje ali bolj desno od ostalih treh (Slika 3.8). Če se, vrstni red oglišč zamaknemo za dve polji. Po teh dveh korakih bo rotacija knjige normalizirana.



Slika 3.8: Levo: Oglišče A je višje od D. Desno: Oglišče A je levo od B.

Knjige vodoravnega formata moramo kameri pokazati obrnjene na bok. Nato jih obravnavamo, kot bi bile navpične. Problem nastane pri kvadratnih knjigah ali knjigah, ki imajo tako majhno razliko med širino in višino, da bi jih lahko zaradi napak v meritvah označili narobe. Ta problem bi lahko rešili le z zahtevo, da nagib knjige ne sme presegati kota

45 stopinj, saj sicer ne moremo oceniti, v katero smer je nagnjena kvadratna knjiga. Žal pa izkušnje med meritvami kažejo, da je kot težko oceniti po občutku in knjigo, ki jo držimo za oglišče, zelo pogosto nagnemo nekoliko predaleč. Ta problem ostaja nerešen in prepoznavanje kvadratnih knjig ni zanesljivo.

Za izračun projekcije potrebujemo parametre a_{ij} v enačbi

$$[x'\ y'\ w'] = [u\ \mathrm{v}\ 1] * \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix}.$$

Koordinate vsake od ciljnih štirih točk predstavljata koordinati u in v. Vrednosti $x', y' \, in \, w'$ so uporabljene v enačbah koordinat vsake od najdenih štirih točk:

$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}}$$

in

$$y = \frac{y'}{w'} = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}}.$$

Parametrov a_{ij} je 8, znani pa so štirje pari x in u ter y in v, torej lahko rešimo linearni sistem enačb. Ko poznamo formulo za preslikave med začetnimi in končnimi oglišči, jih lahko uporabimo tudi na vseh ostalih točkah. Ko poznamo vse parametre, izvedemo preslikavo z zgornjima formulama [*22*].

Pomembna je smer preslikave. Ker nas zanima ciljna tabela in ne izvorna, je pomembno, da za vsako točko v njej najdemo podatke v izvorni tabeli in ne obratno. V slučaju, da bi bila začetna slika manjša od končne, bi razporedili vse dane podatke, a ne bi zapolnili vseh mest v rezultatu. Rezultati množenja bodo decimalna števila, tabele pa so diskretne, torej moramo lokacije interpolirati [*8*]. Uporabimo linearno interpolacijo, ki za podatke, ki ležijo med polji, za vrednost določi uteženo povprečje teh polj. Vpliv vsakega polja je sorazmeren z njegovo bližino točki. Končni rezultat transformacije je naravnana in obrezana knjižna platnica standardnih dimenzij.

3.5 Normalizacija svetlosti slike

Zaradi spremenljivih svetlobnih pogojev je smiselno slike normalizirati po svetlosti. Pri tem si pomagamo s histogramom. Vse točke po svetlosti razdelimo v 256 razredov. Števec za vsak razred delimo z vsoto razredov in množimo z 255. Nato izračunamo določeni integral $H'(i) = \sum_{0 \le j \le i} H(j)$ in popravimo svetlost vsake točke na $\mathrm{nova(x,y)} = \mathrm{H'(stara(x,y))}$ [*17*].

Rezultat je slika, v kateri na svetlost vsake točke vpliva delež slikovnih elementov, ki so v bili začetni sliki temnejši od nje. Področja v histogramu, v katerih je bilo zelo malo točk, so zdaj skrčena. Svetlejše izmed točk s teh področij postanejo še svetlejše, temnejše pa še temnejše. Povprečna svetlost slike se normalizira in hkrati se poveča kontrast (Slika 3.9). Oboje povzroči, da so slike bolj jasne in sistem bolj odporen na razlike v svetlobnih pogojih [*17*]. Tako obdelane slike so vhod za učenje in razpoznavanje.

Slika 3.9: Barvna slika platnice s povečanim kontrastom.

Poglavje 4 Klasifikacija slik

Slika knjige je sedaj naravnana in normalizirana. Zapisati jo moramo v bolj učinkoviti obliki in najti najboljše ujemanje v podatkovni bazi. V tem poglavju bomo predstavili oba koraka. Najprej moramo izrezano sliko povzeti na način, ki ohrani bistvene informacije in je hiter za obdelavo. Za to nalogi bomo preizkusili dve tehniki. V prvi bomo z algoritmom SIFT poiskali lokalne značilke v sliki in izračunali histogram njihovih pojavitev. V drugi pa bomo sliko z algoritmom PCA preslikali v nov koordinatni sistem, ki ga lahko z majhnimi izgubami zapišemo v okrnjeni obliki.

Na teh podatkih bomo učili klasifikatorje, ki med vzorci določenega razreda skušajo najti povezave, ki bodo napovedale razred neznanega novega vzorca. Uporabili bomo klasifikatorja SVM in *k*-NN in preprost klasifikator s predstavnikom.

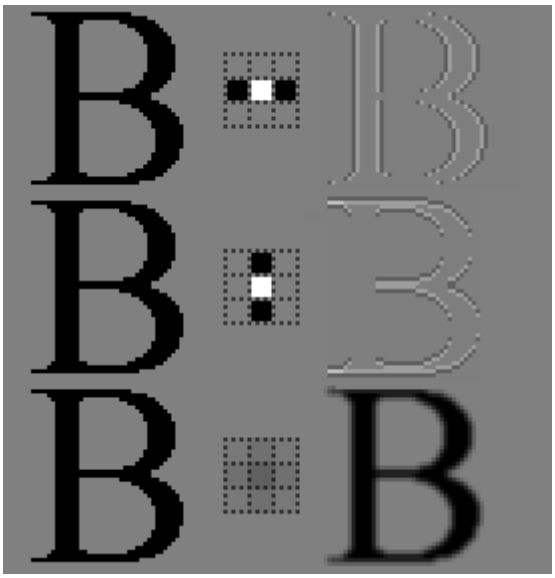
4.1 SIFT

Eden od ključnih problemov računalniškega vida je, da slike vsebujejo ogromno količino informacij. Značilke v računalniškem vidu predstavljajo bistvene informacije, ki jih z nekim algoritmom najdemo v slikah in nam služijo pri reševanju določenih problemov, kot je iskanje ujemajočih se slik.

Značilka bi bila lahko povprečna svetlost ali kontrast slike, a pogosto nam bolje služijo lokalne lastnosti slike. Zanimajo nas lokacije v sliki, ki si delijo nekatere zaželene lastnosti. Pri iskanju ujemanja v slikah si želimo, da bi na podobnih slikah našli podobne ključne točke. Algoritem za iskanje mora biti dovzeten za različne vzorce, a hkrati odporen na naključne spremembe, npr. na svetlost, rotacijo, pozicijo ali perspektivo. Iz tega razloga pri iskanju ključnih točk pogosto dobimo rezultate na območjih z velikimi spremembami v sliki, kot so oglišča predmetov, saj na teh mestih manjše spremembe v pogojih zajema slike nimajo dovolj vpliva. Če posnamemo sliko svinčnika in algoritem izbere kot ključno točko konico svinčnika, pričakujemo, da bo isti algoritem tudi na sliki tega svinčnika pod drugim kotom našel ključno točko na konici svinčnika in bosta značilki, izračunani v teh pozicijah, podobni. Ta algoritem bi lahko našel podobno značilko tudi v sliki radirke, a zgolj po naključju in z manjšo verjetnostjo. Če posnamemo dve sliki svinčnika in dve sliki radirke, statistično pričakujemo boljše ujemanje med značilkami v dveh slikah istega predmeta; seveda pod

pogojem, da smo uporabili primeren algoritem za iskanje značilk in tudi primeren algoritem za primerjavo ujemanja.

Eden najbolj uspešnih algoritmov za iskanje značilk je SIFT. Algoritem SIFT v črno-beli sliki išče ključne točke z velikim kontrastom, njegova bistvena prednost pa je neodvisnost od merila. Čeprav lahko oster kot pod drobnogledom izgleda zaobljen ali zaobljen kot od daleč oster, algoritem SIFT najde območja, ki kljub takšnim spremembam ostanejo prepoznavna in jih opiše na razločen način. Neodvisnost od merila je dosežena tako, da algoritem preišče isto sliko z več merili. Delno je to doseženo s spreminjanjem dejanske velikosti slike, delno pa simulirano. Ustvarimo kopije slike, ki se po velikosti ločijo za faktor 2. Te kopije so imenovane oktave. Izraz izvira iz glasbene teorije, kjer je faktor 2 razlika med frekvencama dveh not s podobnim zvokom, npr. A4 in A5 ali C#2 in C#3. V prvi oktavi se slika poveča za faktor 2 in zamegli, v drugi oktavi je originalna slika, v tretji je slika dvakrat manjša, v četrti pa štirikrat manjša. Navadno so štiri oktave dovolj. Znotraj vsake oktave pa zvezno spreminjanje merila simuliramo tako, da slike zameglimo z uporabo konvolucije [*13*].



Slika 4.1: Primeri konvolucije za iskanje robov in zamegljevanje. Dejanska velikost jeder je 3 x 3 slikovne elemente. Siva barva predstavlja vrednost 0.

Konvolucija (Slika 4.1) je postopek, pri katerem seštejemo zamaknjene kopije signala (v tem primeru slike), pomnožene z določenimi koeficienti, ki tudi sami tvorijo signal. Tabela koeficientov pri različnih zamikih se imenuje konvolucijsko jedro. Če je $A(x, y)$ točka v konvolucijskem jedru dimenzij $w_A \; x \; h_A$ in $B(i, j)$ točka v začetni sliki, se točka $C(i, j)$ v ciljni sliki izračuna kot

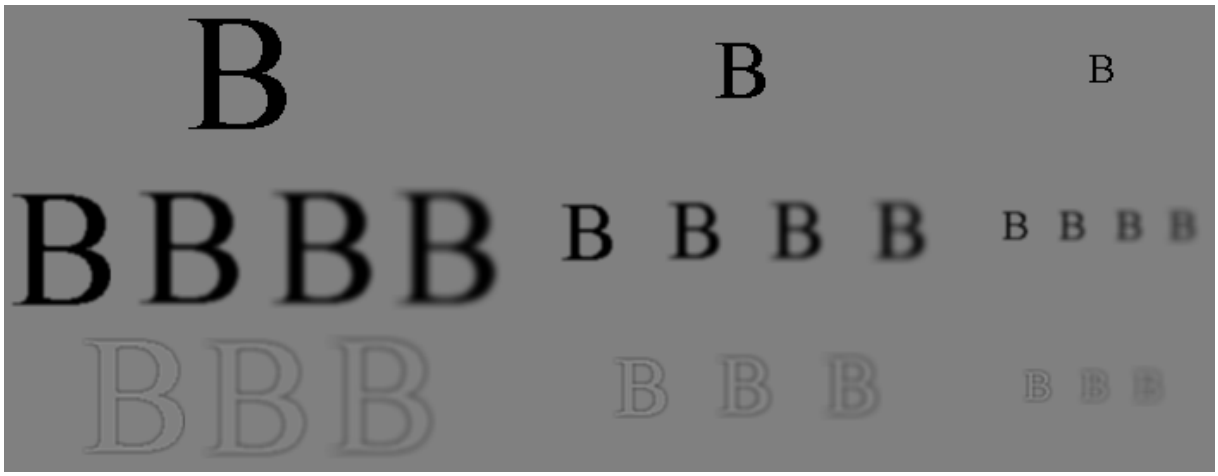
$$C(i,j) = \sum_{x=0}^{w_A-1} \sum_{y=0}^{h_A-1} A(x,y) * B(i-x, j-y).$$

Primer: če seštejemo dve kopiji slike s koeficientom ene polovice in drugo zamaknemo za razdaljo d v levo, to pomeni, da se v vsakem slikovnem elementu zdaj nahaja povprečje tega elementa in elementa, ki je za d levo od njega. Če je bila prej v sliki neka točka (x, y) povsem drugačna od točke (x-d, y), ji zdaj postane podobna; če ji je bila podobna že prej, pa taka ostane. Na ta način konvolucijo uporabimo, da na željenih razdaljah zameglimo razlike. Lahko uporabimo tudi negativni koeficient. Če je bila točka (x, y) povsem drugačna od točke (x-d, y), z odštevanjem ohranimo to razliko; če ji je bila prej podobna, se pri odštevanju rezultat približa ničli. S kombinacijami negativnih in pozitivnih koeficientov v sliki torej izpostavimo željene lastnosti [*9*].

Med vsemi konvolucijskimi jedri je za dobro zameglitev podatkov posebej uporabno Gaussovo jedro. Vsebuje dvodimenzionalno tabelo približka Gaussove ali normalne razporeditve

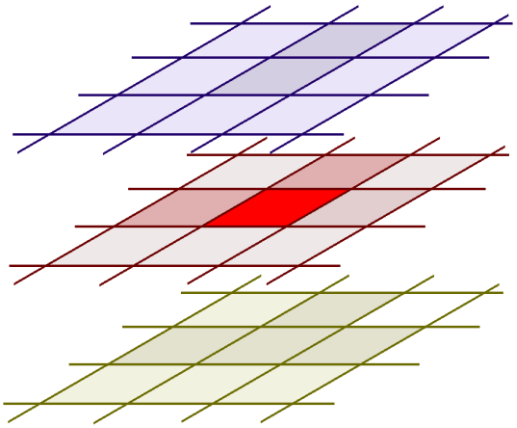
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

ki v statistiki opisuje povprečje večjega števila naključnih razporeditev. Spremenljivka σ predstavlja velikost Gaussovega jedra. Gaussova porazdelitev je zvezna, česar ne moremo zajeti v matriki. Poleg tega je neskončna, a so vrednosti daleč od jedra izjemno majhne in jih za potrebe algoritma SIFT lahko zanemarimo [*9*].



Slika 4.2: Prostor meril (scale space).

Algoritem SIFT išče ključne točke, ki močno izstopajo v različnih merilih. Za vsako možno razdaljo bi torej morali uporabiti konvolucijsko jedro, ki poudari razlike na tej razdalji, vse ostale pa prikrije. Laplaceov operator nam v točki T za dano matematično funkcijo pokaže, kako se njena vrednost spreminja z oddaljenostjo od točke T. S kombinacijo tega operatorja in Gaussove porazdelitve različne velikosti lahko izračunamo konvolucijsko jedro, ki v sliki izpostavi razlike poljubnega merila. Vendar je ta operacija za procesiranje zelo potratna. Kot približek temu operatorju se v algoritmu uporablja preprosto razlika konvolucije z Gaussovim jedrom druge velikosti (Slika 4.2). Znotraj vsake oktave (zgornja vrstica) sliko zameglimo z več Gaussovimi jedri naraščajočih dimenzij (srednja vrstica) in rezultate sosednjih odštejemo (spodnja vrstica). Te tabele nam kažejo na spremembe v sliki v vseh željenih merilih [*13*].



Slika 4.3: Primerjava slikovnega elementa s 26 sosedi v prostoru meril.

V vsaki od teh tabel poiščemo ekstremno velike in majhne vrednosti. Vsak slikovni element primerjamo s sosedi v vseh osmih smereh in še z devetimi sosedi v tabelah nad in pod njo (Slika 4.3). Če je njegova vrednost večja ali manjša od vseh 26 sosedov, je dober kandidat za ključno točko. Lokalne maksimume in minimume je mogoče izračunati celo bolj natančno, kot nam sicer diskretna tabela omogoča. Iz vrednosti sosednjih točk lahko sestavimo matematično funkcijo. Z odvodom te funkcije najdemo lokalni maksimum ali minimum z bistveno višjo natančnostjo. Za vsako najdeno točko si shranimo še merilo, na katerem smo jo našli [*13*].

V prejšnjem koraku smo našli preveč ključnih točk. Najprej zahtevamo določeno mero kontrasta. Kontrast smo že ocenili s tem, ko smo odšteli sosednje tabele, torej preprosto poiščemo vrednost v ključni točki in jo odstranimo, če je premajhna. Poleg tega želimo v sliki najti manj robov in več oglišč. Robove detektiramo tako, da spremljamo spremembe v matriki čez ključno točko v dveh pravokotnih smereh. V ravnini bo kontrast zelo majhen v obeh

smereh, v oglišču pa bo v obeh velik. A na robu bomo našli velik kontrast v smeri, pravokotni na rob, in majhen v smeri robu. Točke, ki ustrezajo temu pogoju, izločimo [*13*].

V naslednjem koraku poskrbimo za neodvisnost značilk od rotacije predmeta v sliki. Če bi posneli dve sliki svinčnika z različnih kotov in našli obakrat ključno točko na konici, a ne bi upoštevali, da je na prvi sliki konica obrnjena v drugo smer, bi iz točk izračunali povsem drugačne značilke. Točkam moramo določiti neko orientacijo, da lahko v postopku kompenziramo te razlike. Za vsako točko $L(x, y)$ v tabeli pri merilu, pri katerem je bila najdena, določimo moč $m(x, y)$in smer $\Theta(x, y)$ spreminjanja gradienta, kot prikazujeta naslednji enačbi [*13*]:

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\Theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-y,y)}\right).$$

Ustvarimo histogram s 36 razredi. Izračunamo orientacije in moči za točko in vse njene sosede in jih razporedimo v histogram. Na podlagi orientacije izberemo enega od razredov, prištejemo pa vrednost, ki je utežena z močjo gradienta in z lokacije vsake točke v Gaussovem oknu s centrom v ključni točki in merilom, ki je 1,5-krat večje od merila ključne točke. V histogramu najdemo maksimalno vrednost in pripadajočo smer določimo kot orientacijo ključne točke. Če je v histogramu več maksimalnih vrednosti, uporabimo prvo od njih, za vse ostale pa ustvarimo nove ključne točke z enakimi koordinatami x in y in merilom σ [*13*].

Našli smo vse ključne točke in jim določili merila in orientacije. Sedaj v teh točkah izračunamo lokalne značilke. Te morajo biti dovolj prepoznavne, da razločimo povsem različne strukture, a hkrati dovolj splošne, da opazimo podobne. Okoli vsake točke pregledamo okno 16 x 16 točk, ki ga razdelimo na 16 oken s 4 x 4 točkami. Znotraj vsakega okna spet izračunamo histogram. Za vsako točko izračunamo orientacijo od katere odštejemo orientacijo ključne točke. S tem dosežemo neodvisnost od rotacije slike. Števcu za razred histograma, v katerega spada orientacija, prištejemo moč, ki jo utežimo z Gaussovim oknom. Iz vseh 16 polj smo izračunali po 8 vrednosti, torej imamo skupaj 128 vrednosti. Vsa polja v histogramu v_i še normaliziramo po enačbi

$$v_i = v_i / \sqrt{\sum_{1 \le j \le 128} {v_j}^2}$$

in za vsako ključno točko smo našli vektor značilk, ki jo opisujejo [*13*].

Število značilk SIFT v sliki je spremenljivo, v kasnejših korakih pa bomo uporabili tehnike strojnega učenja, ki zahtevajo, da so vhodni podatki standardizirani in opis vsakega vzorca sledi enakemu redu. V primeru, da naš algoritem vključuje odštevanje enega vzorca od drugega, morata imeti oba vzorca enako število dimenzij, ki predstavljajo enake lastnosti v enakem vrstnem redu. V ta namen značilke organiziramo v vektorje.

4.1.1 Vreča besed

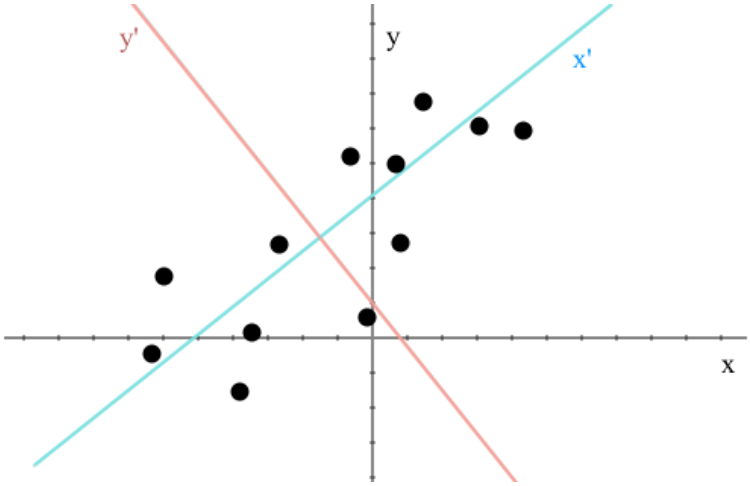
Vreča besed je koncept v umetni inteligenci, ki je bil razvit za analizo besedil. Zanemarimo tako vrstni red besed kot tudi slovnico in v vsakem besedilu (stavku, odstavku, dokumentu...) preštejemo število pojavitev določenih besed. Tudi če ne poznamo strukture besedila, namreč zlahka ločimo besedišče, uporabljeno v pesniški zbirki, od besedišča v medicinskem priročniku [*12*].

Tudi v računalniškem vidu se metoda uporablja za nadaljnje nižanje števila značilk in za normalizacijo njihovega števila. Algoritem SIFT izračuna 128 značilk za vsako ključno točko v sliki, kar je v analizi večjega števila slik lahko odločno preveč. Poleg tega ne vemo vnaprej, koliko ključnih točk bomo našli. Opazimo pa, da so vse ključne točke, najdene na konicah svinčnikov, opisane z zelo podobnimi značilkami. Namesto da bi previdno opisovali vsak svinčnik posebej, je koristen podatek že to, ali slika vsebuje svinčnike ali ne, in če jih, koliko jih je na sliki. Ostra temna konica je torej ena od »besed«, ki jih v slikah iščemo. Pregledamo lahko učno množico in definiramo razrede, med katere razdelimo vse značilke. Slike namesto lokalnih značilk opišemo s histogramom pojavitev različnih tipov značilk [*12*].

Pogosto se za iskanje tipičnih ključnih točk uporablja metoda voditeljev (*k*-means clustering). Cilj te metode je, da dane točke razdeli med *k* voditeljev tako, da minimizira vsoto kvadratnih razdalj med vsako točko in voditeljem razreda, ki mu pripada. Najprej naključno izberemo *k* voditeljev razredov. Izračunamo razdalje vsake točke do vsakega voditelja in jih razporedimo k najbližjim. V vsaki skupini najdemo težišče in ga določimo kot voditelja. Spet razdelimo točke k najbližjim voditeljem in spet popravimo voditelje. Postopek ponavljamo, dokler se razpored točk med voditelje ne neha spreminjati. Takrat seštevek kvadratnih razdalj v vseh razredih doseže lokalni minimum. Razporeditev je spremenljiva glede na naključno razporeditev na začetku. Za boljšo razporeditev algoritem poskusimo večkrat in izberemo najboljšo delitev [*10*].

4.1.2 PCA

Povsem ločeno poskusimo še algoritem PCA. Gre za postopek, s katerim podatke projiciramo v nov koordinatni sistem, ki nam omogoča njihovo skrajšanje s minimalno izgubo natančnosti. Za množico n-dimenzionalnih vzorcev najdemo nov n-dimenzionalni prostor, v katerem so dimenzije urejene po pomembnosti (Slika 4.4). Vzorci se v prvi dimenziji razlikujejo bolj kot v drugi; v drugi se razlikujejo bolj kot v tretji in tako dalje. Ko se bližamo n-ti dimenziji, postane razmik med vzorci v tej dimenziji zelo majhen. Posledično lahko število dimenzij omejimo in vzorce opišemo na krajši način, saj preostale dimenzije ne predstavljajo velikega dela podatkov. Možno je celo, da so zadnje koordinate enake nič, če imamo med podatki konstantne ali popolnoma soodvisne spremenljivke [*11*].



Slika 4.4: Dvodimenzionalni primer prostora PCA

Vse učne primere, na katerih bomo učili algoritem PCA, pretvorimo iz pravokotne tabele slikovnih elementov v raven vektor in jih zložimo enega na drugega v novo matriko. Za vsak stolpec izračunamo povprečno vrednost in jo odštejemo od vseh vrednosti v stolpcu. Iz te matrike M nato izračunamo kovariančno matriko $C = \frac{1}{n-1} M^T \cdot M$. Kovariančna matrika nam za vsak par dimenzij v začetnem prostoru pove, ali sta pozitivno korelirani ali negativno ali sploh ne. Kovariance so absolutne vrednosti, torej so odvisne od obsega vpletenih spremenljivk, a vse naše spremenljivke so svetlosti slikovnih elementov. Te imajo vedno enak obseg in ne zahtevajo dodatne normalizacije [*11*].

Najti moramo matriki V in D v enačbi $V^{-1}CV = D$. Za ta postopek obstaja mnogo učinkovitih algoritmov. Stolpci tabele V predstavljajo lastne vektorje kovariančne matrike C, matrika D pa je diagonalna matrika z lastnimi vrednostmi za vsak lastni vektor v pripadajočem stolpcu in ničlami povsod drugje. Lastni vektorji so enotni in pravokotni drug na drugega. V starem

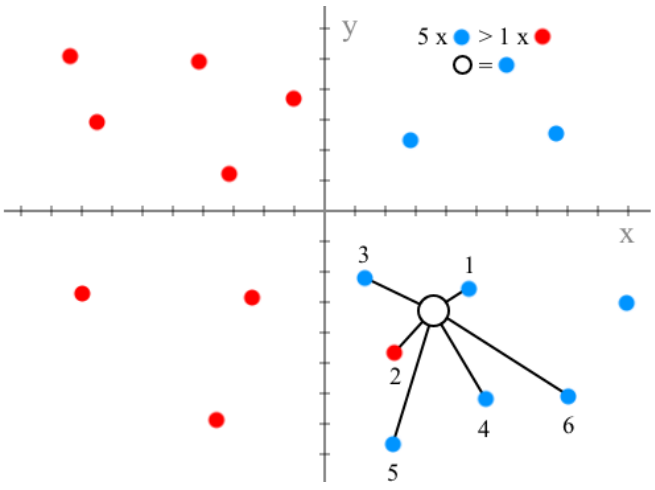
koordinatnem sistemu nam povedo naklon osi novega koordinatnega sistema. Imajo enako število polj kot vhodni podatki in s skalarnim produktom izračunamo dolžino projekcije vsakega vzorca na enega od lastnih vektorjev. Če vektorje naložimo drugega pod drugega po redu velikosti pripadajočih lastnih vrednosti od največje do najmanjše, dobimo preslikavo v koordinatni prostor, v katerem so osi urejene po pomembnosti. Iz kovariančne tabele smo izračunali en lastni vektor za vsako dimenzijo originalnega koordinatnega sistema, a pri oblikovanju preslikave lahko število vektorjev omejimo in skrčimo podatke. Vsak testni vzorec sedaj zapišemo kot stolpično matriko in zmnožimo z izračunano preslikavo, da dobimo vzorec v stisnjeni obliki [*11*].

4.2 Klasifikacija značilk

Oba pristopa nam dasta krajši opis zajete slike. Na njiju bomo preizkusili delovanje več tipov klasifikatorjev, ki jih bomo učili na učni množici z označenimi razredi. Preizkusili bomo klasifikator z evklidsko razdaljo do predstavnika vsakega razreda, klasifikator *k*-NN, ki deluje na podlagi sosednosti vzorcev in klasifikator SVM, ki računa meje v *n*-dimenzionalnem prostoru.

4.2.1 k-NN

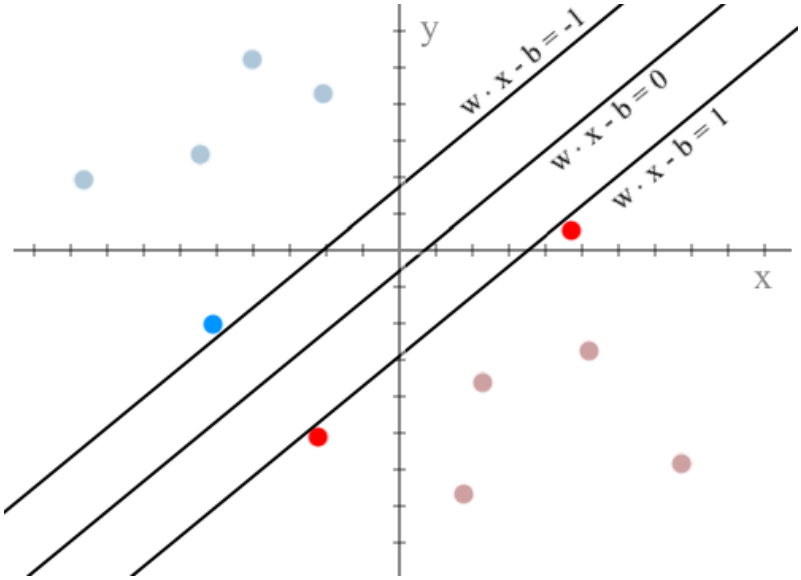
Pri *k*-NN izračunamo *k* najbližjih učnih primerov za vsakega testnega in preštejemo pogostosti različnih razredov (Slika 4.5). Za vsak testni primer izračunamo evklidsko razdaljo $d = \sqrt{\sum_{0 \le i \le n} (učni_i - testni_i)^2}$ do vseh učnih primerov in hranimo tabelo *k* najbližjih skupaj z njihovimi razredi. Izberemo tisti razred, ki se v tabeli pojavi največkrat. Če se več razredov pojavi z enako pogostostjo, sklepamo na podlagi njihove bližine [*7*].



Slika 4.5: Kategorizacija primera z algoritmom *k*-NN.

4.2.2 SVM

Ideja klasifikatorja SVM je, da med podatki potegne meje, ki jih kar se da dobro razdelijo na skupine. Slika 4.6 prikazuje primer delitve dvodimenzionalnih podatkov, kjer podatke loči meja v dvodimenzionalnem prostoru, to je premica. V trodimenzionalnem prostoru bi jih ločila ravnina, v *n*-dimenzionalnem pa hiperravnina. Med vsemi možnimi mejami, ki bi jih lahko potegnili med učnimi vzorci, moramo izbrati čim boljšo. S klasifikacijo vzorcev, ki ležijo daleč od črte, ne bomo imeli težav, zato so bistveni najbližji vzorci, imenovani podporni vektorji (support vectors), saj sta na njih oprti dve hiperravnini, ki definirata mejo z maksimalnim razmakom [*4*]. Osnovni pristop podatke jasno razdeli na dva dela. Kasneje si bomo pogledali, kako dosežemo delitev manj jasnih podatkov in kako razdelimo podatke med več kot dve skupini.



Slika 4.6: Primer delitve dvodimenzionalnih podatkov z optimalno mejo.

Ravnine v *n*-dimenzionalnem prostoru opisuje enačba

$$\vec{w} \cdot \vec{x} - b = \begin{cases} 0, splošna\ enačba, idealna\ meja\ med\ oznakama\ 1\ in\ -1 \\ 1, \quad meja\ oprta\ na\ podporne\ vektorje\ z\ oznako\ y = 1 \\ -1, \quad meja\ oprta\ na\ podporne\ vektorje\ z\ oznako\ y - 1 \end{cases}.$$

V tej enačbi poljuben vzorec predstavljata *n*-dimenzionalen vektor $\vec{x}$ in oznaka za pozitiven ali negativen razred y, ki je uporabljena v klasifikaciji. Vektor $\vec{w}$ je *n*-dimenzionalen koeficient hiperravnine, b pa konstanta. Če skalarni produkt na primer izvedemo na vektorjih dolžine ena ali dve, dobimo znani enačbi premice in ravnine, a naša hiperravnina bo imela toliko dimenzij, kolikor smo jih izračunali v poglavju 4.1. Razdalja d med dvema mejama je

$d = \frac{2}{\|w\|}$, torej moramo minimizirati $\|w\|$ ob pogoju, da vse točke ležijo na svoji strani meje [*4*].

Vendar pa ni nujno, da vse točke res ležijo na svoji strani. Zato uporabimo mero napake za vsako točko $\xi_i = \max(0, 1 - y_i(\vec{w} \cdot \vec{x_i} - b))$. Če točka leži na pravi strani meje za svojo oznako y_i, je rezultat enačbe 0, sicer pa je sorazmeren razdalji do meje. Za klasifikacijo moramo algoritmično optimizirati $\vec{w}$ in b za minimalen rezultat enačbe $r = \frac{w^T w}{2} + C \sum_{i=1}^{N} \xi_i$, v kateri parameter C predstavlja nastavljivo utež. Za klasifikacijo vsakega primera med dve skupini y lahko nato uporabimo formulo $y = sig(\vec{w} \cdot \vec{x} - b)$ [*5*].

Razviti klasifikator še vedno razloči le med dvema razredoma, knjig pa imamo poljubno število. Razširiti ga moramo torej za večrazredno klasifikacijo. Naučimo klasifikatorje SVM za vsako kombinacijo dveh razredov v učni množici, pri čemer izpustimo preostanek učnih primerov. Med klasifikacijo uporabimo glasovanje vseh klasifikatorjev. Ne vemo, v kateri razred bo klasifikator, ki izbira med razredoma A in B, uvrstil knjigo iz razreda C. A to ni bistveno, če se klasifikatorja, ki izbirata med razredoma A in C ter med razredoma B in C, strinjata, da knjiga spada v razred C. Da zagotovimo čim bolj enakomerno razporeditev nesmiselnih primerjav, je potrebno, da so razredi v učni množici uravnovešeni. Če je učnih primerov za razred A bistveno več kot katerih drugih, bo sistem nagnjen h klasifikaciji v razred A [*5*].

4.2.3 Klasifikator s predstavnikom

Preprost klasifikator bi namesto celotne učne množice za klasifikacijo uporabil le po en tipičen primer vsakega razreda. Za vsak testni primer nato izračunamo evklidsko razdaljo do predstavnikov vseh razredov $d = \sqrt{\sum_{0 \le i \le n} (predstavnik_i - testni_i)^2}$. Izberemo tisti razred, ki mu je testni primer najbližje.

Najboljši primer bi lahko izračunamo kot hipotetičen učni vzorec, ki bi bil v danem opisu povprečje vseh ostalih vzorcev v razredu. Lahko pa po neki metriki izberemo v vsakem razredu najboljšega od učnih vzorcev. Za vsak učni vzorec v razredu seštejemo razdalje do vseh ostalih. Kot predstavnika razreda izberemo tisti vzorec, ki je vsem ostalim najbližji. Ta vzorec nam kasneje lahko pomaga tudi pri razlagi rezultatov.

Poglavje 5 Implementacija in uporaba sistema

Pred testiranjem naših algoritmov naredimo pregled uporabljene programske opreme in uporabniškega vmesnika. V prvem delu poglavja si bomo pogledali jezik, knjižnice in razvojno okolje, s pomočjo katerih je bil program ustvarjen. Pri uporabniškem vmesniku bomo simulirali izposojo in vračanje knjig iz knjižnice, poleg tega pa bomo v postopek zajema slike dodali dodatno mero za zavračanje nenatančno izmerjenih okvirjev na podlagi podobnosti zaporednih meritev.

5.1 Uporabljena programska oprema

5.1.1 C++

C++ je programski jezik, ki je bil razvit leta 1979 kot nadgradnja jezika C z dodanim konceptom razredov. C++ omogoča izvajanje večine kode, napisane v jeziku C z le redkimi spremembami. Tako kot C je tudi C++ zelo učinkovit pri sistemskih opravilih. C++ je prevajani jezik, torej se mora napisana koda pred prvim izvajanjem prevesti v strojno kodo, ta pa se zato izvaja bolj učinkovito, kot če bi se program tolmačil sproti. C++ omogoča uporabnikom zelo velik nadzor in v zameno od njih zahteva toliko večjo mero previdnosti. [*1*] C++ je bil izbran za ta projekt, ker je v njem napisana knjižnica OpenCV [*2*].

5.1.2 Knjižnice

Program je bil razvit z uporabo knjižnice OpenCV [*15*] (Open Source Computer Vision Library) 3.0. Knjižnica nudi funkcije za pogoste naloge v računalniškem vidu in strojnem učenju [*2*]. Poleg analize podatkov OpenCV vsebuje tudi funkcije za vizualizacijo in shranjevanje podatkov. Začetnik projekta je bil Intelov center v Rusiji leta 1999, danes pa ga podpira neprofitna fundacija OpenCV.org.

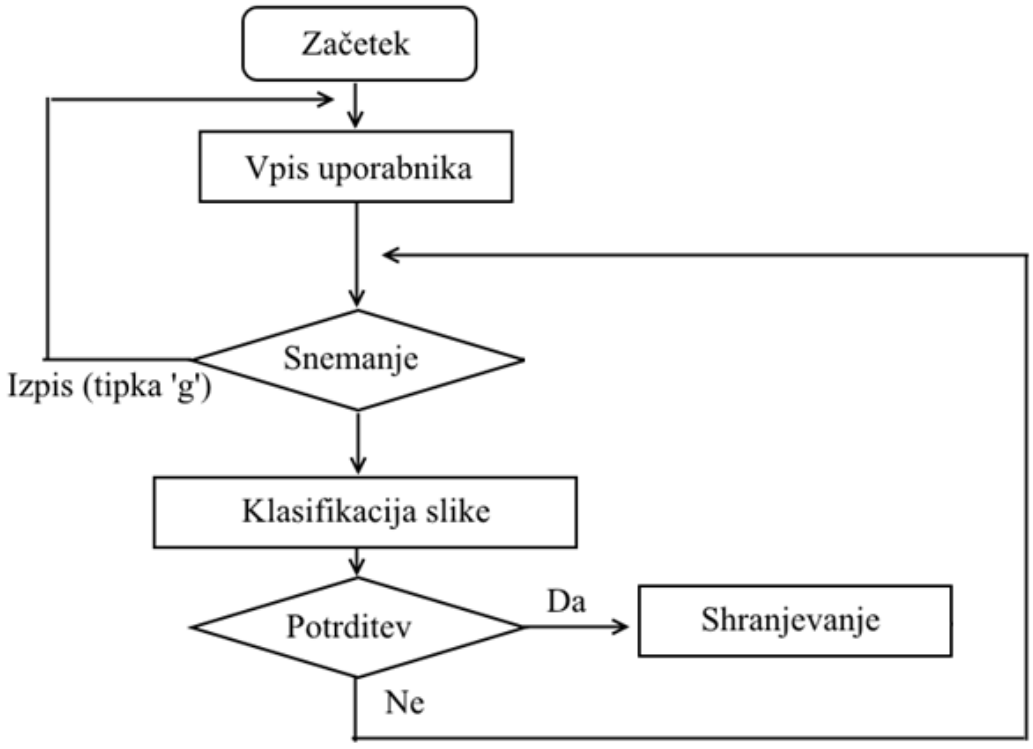
Za delo z datotečnim sistemom in za merjenje časa so bile uporabljene funkcije knjižnice Boost [*3*]. Boost vsebuje učinkovite in prenosljive funkcije za različna področja programskega razvoja, ki so bile v nekaterih primerih celo vključene v C++ standardno knjižnico.

V podporo razvoju projekta so bile uporabljene tudi vizualizacije iz C++ knjižnice Point Cloud Library [*18*] in Java knjižnice Processing [*19*].

5.1.3 Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okolje za operacijski sistem Windows. Visual Studio omogoča razvoj programov v C++ in mnogih drugih jezikih, kot so Visual Basic, C#, Ruby in Python, prirejen pa je tudi za spletni razvoj. Na voljo so grafična orodja za oblikovanje spletnih strani in uporabniških vmesnikov za aplikacije Windows Forms ter orodja za delo s podatkovnimi bazami. Vgrajeni urejevalnik besedila vključuje samodejno dopolnjevanje in obarvanje besedila v skladu s sintakso izbranega jezika ter številne funkcije za lažje programiranje in sodelovanje. Okolje poskrbi tudi za prevajanje kode v izvršljive datoteke in vsebuje razhroščevalnike za vse uporabljene programske jezike.

5.2 Uporaba sistema



Slika 5.1: Diagram poteka programa.

Sistem simulira vpis uporabnika, detekcijo izbrane knjige in njeno izposojo ali vračanje (Slika 5.1). V ozadju so shranjene tri tabele za avtorje, naslove knjig in letnice izida, ena tabela

uporabnikov in tabela trenutnih imetnikov vsake knjige. Ob zagonu uporabnik najprej vpiše svoje ime. Nato program začne detektirati knjige. Za lažjo uporabo program vključuje povratni odziv o trenutni globini knjige. Vse površine, ki se nahajajo izven področja delovanja programa, so zatemnjene. Tako uporabnik lažje poskrbi, da je znotraj področja celotna knjiga, čim manj roke in ničesar drugega.

Za bolj robustno prepoznavanje knjig sistem sproži klasifikacijo šele po več uspešno detektiranih skupinah štirih oglišč, ki jih primerja med seboj. Kadarkoli sistem zazna nov objekt, poišče njegova oglišča in v tabeli hrani zadnjih pet najdenih skupin. Vsakič, ko zazna novo skupino štirih oglišč, preveri razdaljo do vseh skupin v tabeli. Če najde vsaj dve skupini, pri katerih je razdalja med soležnimi oglišči znotraj parametra programa, to pomeni, da je trikrat našel približno iste točke. S tem se z veliko verjetnostjo izognemo naključnim napakam in preprečimo nejasne meritve zaradi gibanja uporabnika. Tipično mora uporabnik knjigo držati na mestu do 0,5 sekunde, preden program najde dovolj enakih oglišč. Po detekciji program v ločenem oknu pokaže rezultat transformacije, iz katerega je vidno, če je pri detekciji oglišč prišlo do napake. Program neha sprejemati okvirje, medtem ko se iz transformirane slike izračunajo značilke in izvede klasifikacija. Izbrani algoritmi in drugi parametri programa se ob vsakem zagonu preberejo iz tekstovne datoteke.

Slika 5.2: Potrditev izbire.

Ko se klasifikacija zaključi, se na zaslonu izpiše predviden naslov knjige skupaj z avtorjem in letom izida, program pa čaka na potrditev uporabnika. Na zaslonu se pojavita dva kvadratna okvirja, levi za pritrdilni odgovor in desni za nikalnega. Uporabnik se s knjigo ali roko dotakne enega od njiju (Slika 5.2) za eno sekundo in program registrira njegov odgovor. Za preprečevanje napak se zahteva, da se uporabnik dotika le enega od kvadratov. Poleg tega se odštevanje ne prične, če je uporabnik držal roko ali knjigo v polju, še preden se je pojavilo. V tem primeru mora polje izprazniti in še enkrat izbrati. Kot grafični prikaz odštevanja se kvadratni okvir v eni sekundi napolni. Če je uporabnik odgovoril pritrdilno, si program zapiše njegovo izbiro. Če je knjiga prosta, mu jo posodi, če si jo je ta uporabnik izposodil, jo sprosti, če pa si jo je izposodil nekdo drug, ga o tem obvesti.

Poglavje 6 Eksperimentalni rezultati

Za oceno delovanja sistema moramo statistično oceniti njegovo natančnost in optimizirati nekatere parametre. Pri algoritmih za iskanje značilk v sliki nas zanima, koliko mest potrebujejo za zanesljiv opis knjige. Z manjšim številom mest namreč lahko pospešimo delovanje, možna pa je celo večja natančnost. Pripravili bomo testno in učno množico posnetkov knjižnih platnic in na njih učili vse kombinacije algoritma PCA in histograma pojavitev značilk SIFT s klasifikatorji *k*-NN, SVM in klasifikatorjem s predstavnikom. Preverili bomo tudi vpliv normalizacije svetlosti in hitrost delovanja vseh uporabljeni algoritmov od omejevanja po globini do klasifikacije. Na koncu bomo ocenili, katera kombinacija algoritmov je po vseh teh kriterijih najbolj primerna za uporabo.

6.1 Postopek merjenja

Za testiranje natančnosti sistema je bilo izbranih 24 knjig različnih dimenzij (Slika 6.1). Za vsako je bilo zajetih med 75 in 120 rektificiranih posnetkov platnic. Slike so bile posnete v petih ločenih snemanjih. 80% vseh slik je bilo posnetih podnevi s kamero Kinect, postavljeno ob oknu, obrnjeno v nasprotno smer. Petina posnetkov je bila narejena v istem prostoru zvečer pri umetni svetlobi. Vsaka knjiga je bila posneta z različnimi pozicijami dlani z vsaj sedmih različnih kotov podnevi in s treh kotov zvečer.

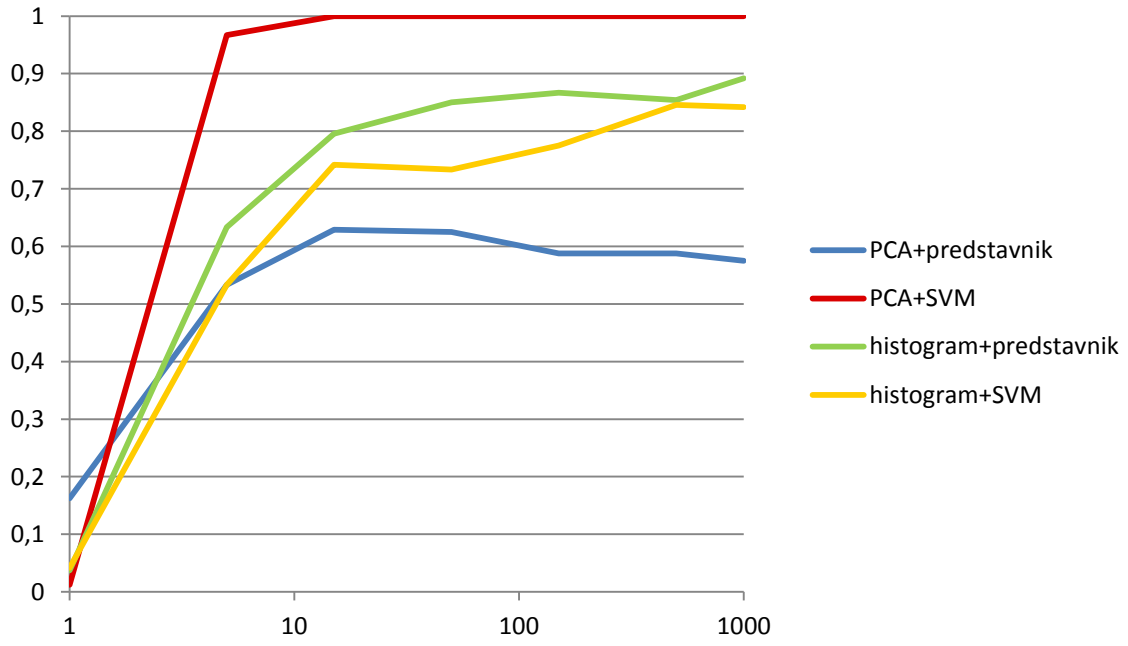
Za večjo raznolikost podatkov znotraj obeh množic so bili podatki z različnih snemanj naključno razdeljeni med učno in testno množico. Za hitrejše procesiranje in preprostejše izračune je bilo za učno množico izbranih 50 naključnih posnetkov vsake knjige, za testno pa 10. Prvi testi so bili narejeni brez uravnavanja svetlosti.

Slika 6.1: Barvne slike predstavnikov vseh razredov in celotna testna množica enega od njih.

6.2 Velikost vektorjev za opis vhodne množice

Izmerimo spreminjanje klasifikacijske natančnosti glede na velikost opisa vhodne množice (Slika 6.2). Na splošno je ujemanje slik, stisnjenih z algoritmom PCA, zelo odvisno od njihove poravnave. Ker so naše slike dovolj natančno popravljene, se PCA odlično izkaže že pri zelo nizkem številu dimenzij. Še več, primerjava s predstavnikom vsakega razreda kaže, da nadaljnje dimenzije vsebujejo veliko šuma. Kompresija PCA namreč nove koordinatne osi po pomembnosti samo prerazporedi, a jih ne uteži. Klasifikator s primerjavo s predstavnikom vsakega razreda upošteva vse te dodatne podatke pri izračunu evklidske razdalje in vsak dodaten podatek poveča razdaljo na nepredvidljiv način. Klasifikator SVM pa lahko poljubno dimenzijo zanemari tako, da izbere hiperravnino, ki je s to osjo vzporedna.

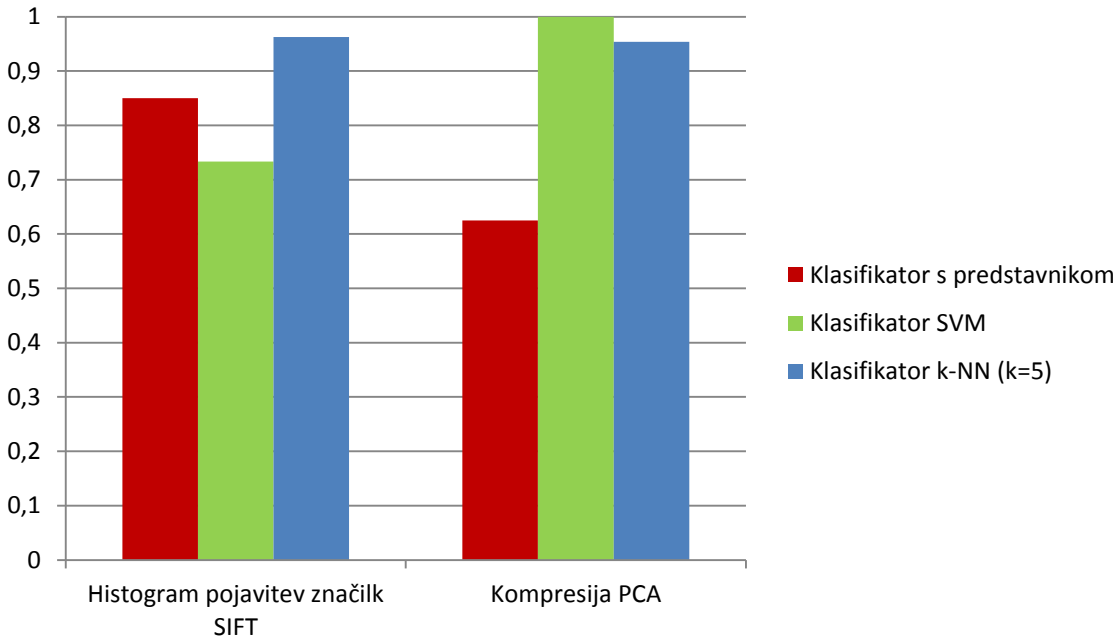
Natančnost opisa s histogramom značilk je manj stabilna. Vidimo, da klasifikator s predstavnikom pri 500 značilkah deluje manj natančno kot pri 150 ali 1000. Klasifikator SVM pa je manj natančen pri 150 kot pri 50 in spet manj natančen pri 1000 kot pri 500. Pri tem ne gre za razliko v detekciji značilk SIFT, saj so bile vsakič uporabljene iste značilke. Povzročitelj napake je torej algoritem, ki kategorizira značilke. Povsem možno je, da z večjo število kategorij podatke po nepotrebnem razpršimo, če so kategorije slabše izbrane. Vsi nadaljnji testi so bili narejeni s 50 dimenzijami.



Slika 6.2: Klasifikacijska natančnost v odvisnosti od števila dimenzij vhodne množice.

6.3 Primerjava različnih algoritmov

Slika 6.3 predstavlja rezultate testiranja različnih kombinacij klasifikatorjev in tehnik za iskanje značilk na slikah brez popravljanja osvetlitve. Vidno je, da kombinacija algoritmov PCA in SVM prevlada, *k*-NN je stabilen ne glede na vrsto značilk, klasifikator s predstavnikom pa je na histogramu bistveno boljši.

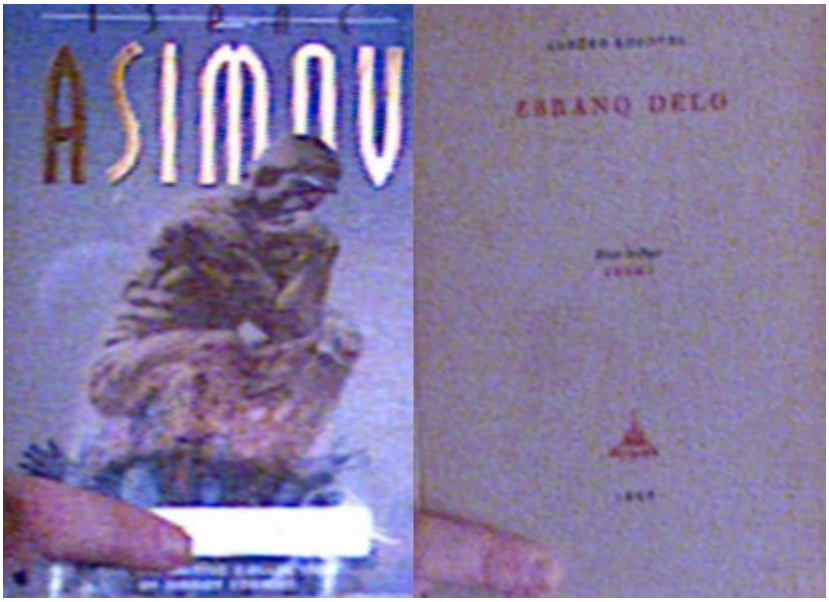


Slika 6.3: Klasifikacijska natančnost na slikah brez popravljanja osvetlitve.

Klasifikator s predstavnikom je v kombinaciji s kompresijo PCA najslabši, a se obnese bolje s histogramom pojavitev značilk SIFT. Dva razloga za to sta bila že omenjena: šum v kasnejših dimenzijah PCA in poravnava slik. Ker so značilke SIFT neodvisne od lokacije, zamik v sliki ni tako kritičen za klasifikator, ki se opira na posamezne primere. A na slikah, stisnjenih z algoritmom PCA, so vir podatkov slikovni elementi in njihova razmerja in s premikom ploskve se vse vrednosti spremenijo. Občutljivost na obrez slike je posebej problematična na manjših knjigah, saj algoritem za iskanje oglišč na njih naredi največje napake. Tudi šum je na manjših slikah bolj izpostavljen, saj so približane. Poleg tega klasifikator brez popravljanja svetlosti ne more prepoznati podobnih vzorcev. Tretji razlog za slabše delovanje z algoritmom PCA pa je občutljivost na spremembe v svetlosti, saj sta si brez uravnavanja svetlosti sliki po evklidske razdalji zelo tuji (Slika 6.4). Večina slik je bila posnetih podnevi, zato je bil idealni primer vedno izbran med njimi in vse slike, posnete zvečer, so bile na videz bolj podobne temnejšim slikam, posnetim podnevi.

Slika 6.4: Primerjava temne in nenatančno detektirane slike z idealno.

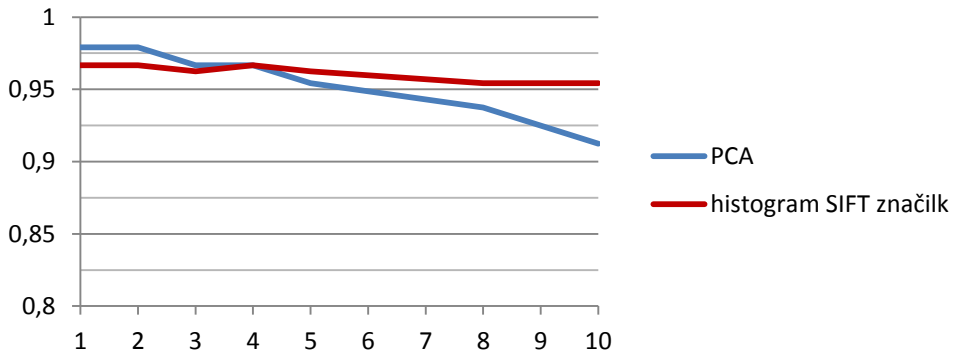
Kombinacija klasifikatorja SVM in kompresije PCA je najbolj natančna izmed testiranih metod, saj dosega pravilno prepoznavanje vseh 100 testnih primerov. Pregled učnih primerov kaže, da je klasifikator sicer manj zanesljiv na preprostih platnicah, kjer je možno najti manj značilk in nekatere od njih niso zanesljive. Poleg tega je histogram normaliziran, zaradi česar se napake v številu značilk povečajo, kadar jih je premalo. Težave ima tudi na odsevnih platnicah, ki so zelo občutljive na spremembe v svetlosti. V podatkih, kompresiranih z algoritmom PCA, je razvidno, da določena področja primerov ne ločijo dobro, zato sistem dimenzij, odvisnih od teh slikovnih elementov, ne bi upošteval. A algoritem SIFT na nepredvidljivih področjih vedno najde različne značilke. Slika 6.5 kaže napis »ASIMOV«, ki je v resnici enobarven, a se sredina sveti v soncu in izgleda svetlejša od ozadja, črka »A« pa je ostala temna. Poleg nje je primer platnice, v kateri je zelo težko najti večje število značilk.



Slika 6.5: Odsevna platnica in pusta platnica.

Klasifikator k-NN se obnese zanesljivo na obeh množicah. Ne zmotijo ga osameli ekstremni primeri v učni množici, če so sicer vzorci združeni v skupine. Pri nobenem primeru nima posebnih težav s prepoznavanjem, niti ni nagnjen k določenim razredom.

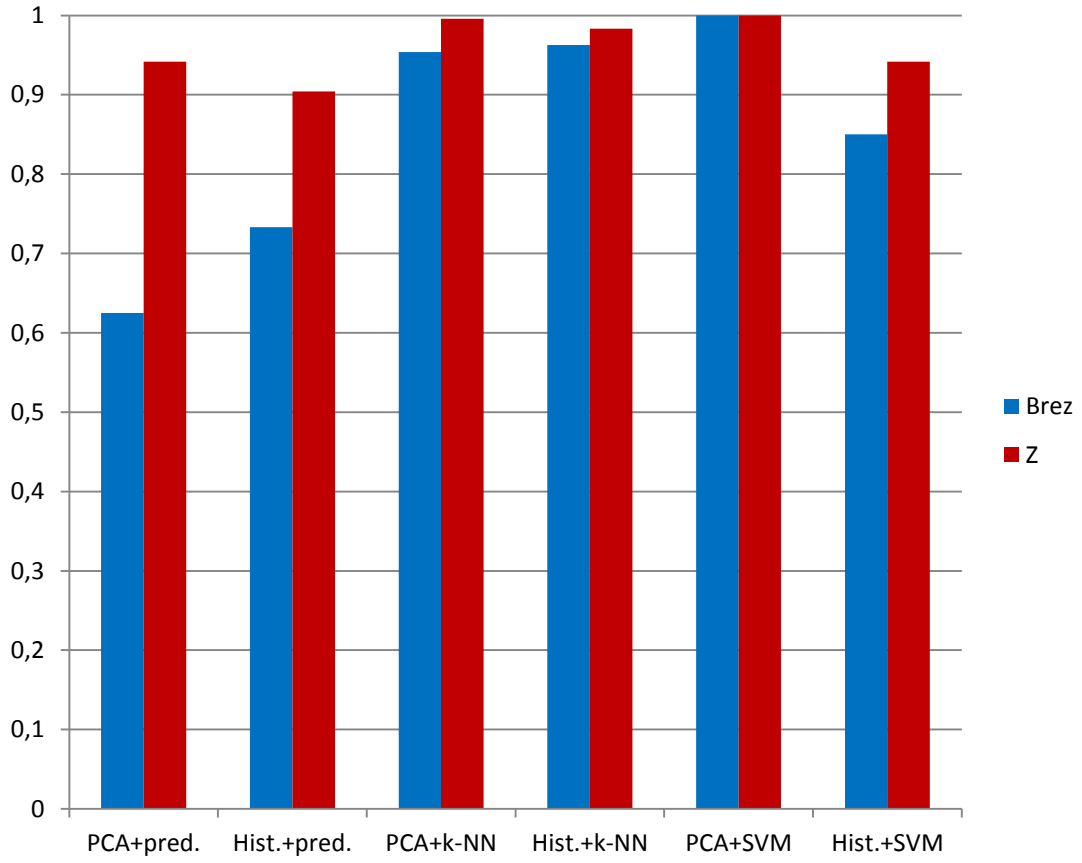
Vendar pa se pri interpretaciji meritev klasifikatorja *k*-NN pokažejo verjetne pomanjkljivosti v načinu zbiranja podatkov. Za večjo pestrost znotraj učne in testne množice so bili posnetki z več snemanj v različnih pogojih združeni in naključno razporejeni med množici. A ker je bila vsaka knjiga večkrat posneta z vsakega kota, ima klasifikator k-NN v učni množici na voljo sosede, ki so nerealno blizu testiranemu posnetku. Njegova natančnost se s spreminjanjem parametra *k* na histogramu značilk ne spreminja bistveno, a na podatkih, stisnjenih z algoritmom PCA kaže padajoč trend, torej so podatki slabše združeni v gruče (Slika 6.6).



Slika 6.6: Graf natančnosti *k*-NN pri različnih vrednostih *k*.

6.4 Normalizacija svetlosti

Slika 6.7 prikazuje izboljšavo rezultatov z dodatkom normalizacije svetlobe. Klasifikatorja s predstavnikom in *k*-NN dosegata večje izboljšave v kombinaciji s kompresijo PCA kot histogramom značilk SIFT. To je logično, saj algoritem SIFT vključuje delno zaščito pred spremembami v osvetlitvi, kompresija PCA pa razlike ohrani oziroma celo poslabša. Prva dimenzija projiciranega prostora je zelo odvisna od svetlosti podatkov, zato lahko namesto normalizacije delno izboljšavo dosežemo tudi tako, da zavržemo prvo dimenzijo.



Slika 6.7: Primerjava klasifikatorjev z vključenim popravljanjem svetlosti.

6.5 Merjenje hitrosti

Za konec preverimo še hitrost delovanja omenjenih algoritmov. Pripraviti moramo algoritme za iskanje značilk. Celotno učno množico želimo skrčiti na 50 dimenzij. Za histogram pojavitev značilk SIFT moramo najprej naložiti 1200 slik in v njih prepoznati značilke SIFT, nato pa izgraditi slovar značilk in ga shraniti. Celoten postopek traja 2 uri in 3 minute, od tega

1 uro in 28 minut traja izgradnja slovarja razredov značilk SIFT. Izračun projekcije PCA je hitrejši in traja 24 minut.

A bolj pomembna je hitrost delovanja programa pri prepoznavanju okvirjev. Izvedli smo 30 do 50 prepoznavanj z vsako kombinacijo algoritmov za iskanje značilk in klasifikacijo ter ocenili povprečno trajanje vsakega koraka.

Tabela 1vsebuje zajete meritve hitrosti različnega dela detekcije knjižne platnice. Največji del splošne priprave podatkov predstavlja algoritem RANSAC. Tega moremo pospešiti le z omejevanjem vhodnih podatkov, s čimer pa bi zmanjšali natančnost detekcije. Normalizacija svetlosti ne zmoti uporabnika in je glede na njen vpliv na klasifikacijsko natančnost nekaterih kombinacij algoritmov dovolj hitra. Pri kompresiji se izkaže, da je histogram pojavitev značilk SIFT izjemno počasen v primerjavi s projekcijo PCA. Projekcija se namreč izvaja z matričnim množenjem, ki je močno optimizirana metoda. Algoritem SIFT je relativno kompleksen in deluje opazno počasneje. Postopek se sicer izvede le enkrat za vsaka zanesljivo detektirana oglišča, a sekunda zamaka je za uporabnika moteča. Tudi po času izvajanja je torej kombinacija projekcije PCA in klasifikatorja SVM odlična.

Postopek	Trajanje	Postopek	Trajanje
RANSAC	114,5 ms	detekcija značilk SIFT	490,8 ms
iskanje oglišč	1,2 ms	računanje histograma značilk	582,1 ms
izrez in transformacija	12,4 ms	kompresija PCA	22,0 ms
konverzija v črno-belo sliko	0,6 ms	klasifikator SVM	0,3 ms
normalizacija svetlosti	16,2 ms	klasifikator *k*-NN	0,6 ms
		klasifikator s predstavnikom	0,2 ms

Tabela 1: Trajanje različnih postopkov med razpoznavanjem.

Poglavje 7 Sklep

V diplomskem delu smo razvili delujoč sistem za prepoznavanje knjižnih platnic. Spoznali smo se z delovanjem globinskega senzorja Microsoft Kinect. Predstavili smo njegove prednosti in slabosti. Prednosti so povsem novi načini iskanja informacij v sliki s filtriranjem trodimenzionalnih podatkov. Pogledali pa smo tudi možne razloge, da v teh meritvah pride do napak. Izpostavili smo razlike v perspektivi in rešitev s procesom registracije.

Implementirali smo postopek za iskanje knjižne platnice v globinski sliki. Uporabili smo nekaj metod za omejevanje količine vhodnih podatkov za algoritem RANSAC, ki z naključnim vzorčenjem v sprejetem okvirju najde ravnino. Razvili smo originalen algoritem za iskanje oglišč nagnjenega pravokotnika v projekciji na ravnino. Ta s postopnim preiskovanjem kotov in daljic najde najbolj oddaljene točke v polju, najdenem v prejšnjem koraku. Ključna motnja v iskanju oglišč je bila uporabnikova roka. Ker je del nje odstranjen v prvem koraku, smo uporabili spremembo težišča kot informacijo o lokaciji preostanka in našli četrto oglišče knjige z drugo metodo. Končno smo izračunali projekcijo iz najdenih štirih oglišč v ravnino kamere, s čimer smo naravnali knjigo.

Pred klasifikacijo smo preizkusili dve metodi za nižanje dimenzionalnosti podatkov za klasifikacijo: projekcijo PCA in histogram pojavitev SIFT značilk. Projekcija PCA podatke preslika v koordinatni prostor z novimi osmi, ki jih izračuna in uredi po pomembnosti v razlikovanju podatkov. Preučili smo delovanje algoritma SIFT za iskanje lokalnih značilk v sliki, ki so odporne na spremembe v velikosti, svetlosti, rotaciji in lokaciji.

Testirali smo klasifikacijo na 24 knjigah. Pregledali smo algoritma SVM in *k*-NN in dodali preprosto iskanje na podlagi razlike glede na predstavnike vsakega razreda. Izmerili smo njihove hitrosti in natančnosti v kombinaciji z različnimi vhodnimi podatki in preizkusili vpliv normalizacije svetlosti. Največjo klasifikacijsko natančnost je dosegel pristop s kompresijo PCA in klasifikatorjem SVM, ki je z zelo omejenim naborom vhodnih podatkov dosegel stoodstotno natančnost.

Pred dejansko implementacijo v pametni knjižnici bi bilo potrebno delovanje sistema testirati na bistveno večjem naboru podatkov. Poleg tega je pristop s prepoznavanjem platnic neprimeren, če želimo slediti posameznim izvodom iste knjige ali če imamo v zbirki več knjig z identičnimi platnicami. A namesto izposoje in vračanja bi bil sistem lahko v knjižnici

uporabljen tudi drugače. Uporabljen bi bil lahko kot pametni vodič, ki bi bralcu povedal več o knjigi in njenem avtorju in mu priporočil podobna dela. Poleg knjig pa bi bil sistem uporaben

tudi za prepoznavanje drugih slik. V prodajalnah z grafičnimi izdelki bi bilo zaželeno, da na izdelek ne bi bilo potrebno lepiti črtne kode.

Nerešen je ostal problem kvadratnih knjig, za katerega bi potrebovali bolj zanesljivo kamero ali algoritem za iskanje oglišč, da bi zagotovili stoodstotno natančno ločevanje kvadrata in pravokotnika. Vredno bi bilo preizkusiti tudi histogram pojavitev drugih deskriptorjev in druge klasifikatorje. Poleg tega sistem na zelo poenostavljen način simulira podatkovno bazo in identifikacijo uporabnika. Za slednje bi bil nadvse primeren sistem s prepoznavanjem uporabnikovega obraza.

Za bolj natančno statistično oceno sistema bi morali ponoviti snemanje testne množice v drugačnih razmerah, da preprečimo nerealno ujemanje s učno množico. Sistem je bil testiran v različnih svetlobnih pogojih in implementirano je bilo tudi popravljanje osvetlitve z normalizacijo svetlobnega histograma. A tudi v praksi z izvajanjem klasifikacije v živo je sistem preprost, zanesljiv in hiter.

Literatura

[1] A Brief Description - C++ Information. [Online]. http://www.cplusplus.com/info/description/ [Dostopano 14.3.2016].

[2] About | OpenCV. [Online]. http://opencv.org/about.html [Dostopano 14.3.2016].

[3] Boost C++ Libraries. [Online]. http://www.boost.org/ [Dostopano 14.3.2016].

[4] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, št. 2, zv. 2, str. 121-167, 1998.

[5] C.C. Chang in C.J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology*, št. 2, zv. 3, 2011.

[6] Coordinate spaces - Microsoft developer network. [Online]. https://msdn.microsoft.com/en-us/library/hh973078.aspx [Dostopano 14.3.2016].

[7] T. Hart in P. Cover, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, št. 13, zv. 1, str. 21-27, 1967.

[8] Geometric image transformations - OpenCV 2.4.12.0 documenation. [Online]. http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html [Dostopano 14.3.2016].

[9] R.C. Woods in R.E. Gonzalez, *Digital Image Processing (3rd Edition)*. New Jersey: Prentice-Hall, Inc., 2006.

[10] J.MacQueen, "Some methods for classification and analysis of multivariate observations," v *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley: University of California Press, 1967, str. 281-297.

[11] I.T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 2002.

[12] M.T. Thorne, N. Cord in M. Law, "Bag-of-Words Image Representation Key Ideas and Further Insight," v *Fusion in Computer Vision: Understanding Complex Visual Content*, 2014.

[13] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, št. 60, zv. 2, str. 91-110, 2004.

[14] Microsoft. Kinect for Windows Sensor Components and Specifications. [Online]. https://msdn.microsoft.com/en-us/library/jj131033.aspx?f=255&MSPPError=-2147217396 [Dostopano 14.3.2016].

[15] OpenCV | OpenCV. [Online]. http://opencv.org/ [Dostopano 14.3.2016].

[16] OpenKinect Protocol Documentation. [Online]. https://openkinect.org/wiki/Protocol_Documentation [Dostopano 14.3.2016].

[17] O. Patel, Y.P.S. Maravi in S. Sharma, "A comparative study of histogram equalization based image enhancement techniques for brightness preservation and contrast enhancement," *Signal & Image Processing : An International Journal (SIPIJ)*, 2013.

[18] PCL - Point Cloud Library. [Online]. http://pointclouds.org/ [Dostopano 14.3.2016].

[19] Processing.org. [Online]. https://processing.org/ [Dostopano 14.3.2016].

[20] M.A. Fischler in R.C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, št. 24, zv. 6, str. 381-395, 1981.

[21] A.J. Tromba in J.E. Marsden, *Vector calculus*. San Francisco: Freeman, 1981.

[22] G. Wolberg, *Digital Image Warping*. Los Alamitos: IEEE Computer Society Press, 1994.

[23] Z. Zhang, "Microsoft Kinect Sensor and its effect," *IEEE MultiMedia,* št. 19, zv. 2, str. 4-10, 2012.